SPACE OBJECT IDENTIFICATION
Using Spatio-Temporal Pattern Recognition

THESIS
Gary W. Brandstrom
Captain, USAF

AFIT/GSO/ENG/95D-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSO/ENG/95D-01

SPACE OBJECT IDENTIFICATION
Using Spatio-Temporal Pattern Recognition

THESIS
Gary W. Brandstrom
Captain, USAF

AFIT/GSO/ENG/95D-01

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

AFIT/GSO/ENG/95D-01

SPACE OBJECT IDENTIFICATION

Using Spatio-Temporal Pattern Recognition

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Gary W. Brandstrom, B.S. Geology and M.S. Geology

Captain, USAF

December 1995

Table of Contents

List of Figures

List of Tables

Abstract

This thesis is part of a research effort to investigate techniques to automate the task of characterizing space objects or satellites based on a sequence of images. The goal is to detect space object anomalies. Two algorithms are considered — the feature space trajectory neural network (FST NN) and hidden Markov model (HMM) classifier. The FST NN was first presented by Leonard Neiberg and David P. Casasent in 1994 as a target identification tool. Kenneth H. Fielding and Dennis W. Ruck recently applied the hidden Markov model classifier to a 3D moving light display identification problem and a target recognition problem, using time history information to improve classification results. Time sequenced images produced by a simulation program are used for developing and testing the anomaly detection algorithms. Two data sets are tested. The first consists of 50 training and 50 test sequences, with 20 images per sequence. The second set contains 23 training and 46 test sequences. Tests on the second data set achieve 100% anomaly detection with no false alarms. The samples in the first data set are more widely spaced making it more difficult to classify, with the best test achieving a 5% false alarm rate with 100% anomaly detection. A variety of features are tested for this problem. Features are derived from the two dimensional (2D) discrete Fourier transform (DFT) with various normalization schemes applied. The FST NN is found to be more robust than the HMM classifier. Both algorithms are capable of achieving perfect classification, but when shot noise is added to the images or when the image sample spacing is increased, the FST NN continues to perform well while the HMM performance declines. A new test is presented that measures how well a test sequence matches other sequences in the database. The FST NN is based strictly on feature space distance; but if the order of the sequence is important, the new test is useful.

SPACE OBJECT IDENTIFICATION

Using Spatio-Temporal Pattern Recognition

I. Introduction

1.1 Nature of the Problem

Space Object Identification (SOI) is one of the tasks performed by Air Force space surveillance sites. SOI answers the following questions about man made space objects: (1) Where is it? This is a function of orbital dynamics; (2) What is its behavior? This addresses local dynamics, like spin, wobble, antenna and panel position; (3) What is it? This is an analysis of size, shape, identification; and (4) What is its capability? This is determined based on the previous three questions [19]. This thesis focuses primarily on "what is it," and "what is its behavior," but may be useful in assessing capability also.

Currently, the task of identifying and analyzing space objects is performed by trained analysts. Identifying space objects is generally not a problem since the object is usually known by its orbital parameters before the images are obtained. The characterization of space object behavior is a more difficult problem. Detection of unusual behavior or appearance (anomalies) is dependent on the analyst's experience.

1.1.1 Problem Statement. Explore spatio-temporal pattern recognition techniques, specifically, the feature space trajectory neural network (FST NN) [12, 13] and hidden Markov model (HMM) algorithms, for detecting anomalous satellite behavior.

1.2 Scope

- Simulated data generated
  - Low earth, sun-synchronous orbit

  - Nadir pointing satellite (this means straight down towards the Earth)
  - Sequences of 20 images spaced 8 to 10 seconds apart
  - Poisson simulated noise
- Features extracted
  - Wedge sampled 2D discrete Fourier transform (DFT)
    * 8 vs 16 wedges
    * With or without dc (Dc is the zero frequency coefficient or total energy cell. It is the middle value of the 2D DFT and is not counted in any wedges)
    * Normalized or not
  - Block of Fourier coefficients
    * 18 low frequency coefficients
    * Normalized or not
    * Energy vs statistical normalization
- Algorithms explored
  - FST NN
    * Distance test
    * Sequence test
  - HMM
    * Ergodic and left-right models
    * Continuous state distributions
    * One and four Gaussian mixtures per state
    * 1, 2, 5, 10 and 15 state models

1.3 Approach

If a satellite looked the same every time it passed overhead or if there were a finite number of viewing angles, anomalous satellite behavior could be detected by simply checking for a matching image sequence. Since this is not the case, a nearest neighbor or statistical type of classifier is appropriate.

Leonard Neiberg and David. P. Casasent have developed a new technique for estimating the pose of an object from 2D images [12]. This technique is called the feature space trajectory (FST) neural network (NN). This technique is very similar to a nearest neighbor classifier, but performs better because it uses the distance to the nearest trajectory instead of nearest neighbor. Neiberg and Casasent used the FST NN to classify images of military vehicles. In order to classify a sequence rather than a single image, two variations of this algorithm are presented. There is the distance test, which simply sums the feature space distance of the images in the sequence, and the sequence test which measures how well the test sequence matches the order of the nearest training sequence.

Kenneth Fielding and Dennis Ruck [6] have recently used Hidden Markov Models (HMM) to classify images of 3D objects based on the way features change with viewing angle over time. This is a statistical method that classifies a sequence based on the probability that a test sequence was produced by a given model. This technique should also be able to classify a sequence of images as normal or anomalous.

1.4 Organization of Thesis

The second chapter of this thesis provides a basic introduction to space object identification (SOI) and orbital mechanics in addition to background information on traditional pattern recognition and the algorithms used in this research. Chapter III covers the methodology, describing the simulated data that were used for the experiments in this thesis, how they differs from real data, and some pre-processing that may be helpful on real data. Chapter III also covers the feature selection and specific classification algorithms tested for this thesis. The fourth chapter shows the results of the tests with a full analysis. The final chapter contains

concluding remarks suggesting possible areas where this research may be applied and areas where further research could be pursued in spatio-temporal pattern recognition.

II. Background

This chapter provides background information to help the reader understand this thesis. Some basic orbital mechanics is presented to help the reader understand normal satellite behavior and the orbital terminology used in this thesis. Next, some background information is provided referencing the traditional pattern recognition techniques from which most pattern recognition algorithms have evolved. The last sections of this chapter cover the feature space trajectory neural network (FST NN), and hidden Markov model (HMM) algorithms to show the reader the origin of the ideas that are applied to the SOI problem in Chapters III and IV.

2.1 Basic Orbital Mechanics

One of the first facts discovered about orbits is that they trace an elliptical path with the Earth at one focus. This comes from Kepler's First Law which states that the orbit of each planet is an ellipse with the Sun at the focus, but this can be applied to satellite orbits as well. Another important fact about orbits is that the orbit plane does not rotate with the Earth, but remains in a fixed orientation in space, except for perturbations such as the force caused by the oblateness of the Earth.

An orbit can be completely and accurately described either with the classical orbital elements, or with a position and velocity vector. Both of these methods are based on a quasi-inertial reference system with the Earth at the center. The principle direction (x-axis) is along the intersection of the Earth's equatorial plane and the Earth - Sun orbit plane, also known as the ecliptic. This principle direction is also known as the vernal equinox or first point of Aries [2] (Figure 1).

The classical orbital elements include two angles that define the plane, four measurements that define the ellipse, and a time which is determined by the satellite's position on the ellipse. The two angles are **right ascension** (Ω), which locates the orbit plane with reference to the vernal equinox and **inclination (i)**, which is the angle between the equatorial plane and orbital plane. The ellipse is defined by a **semimajor axis** (size), **eccentricity** (shape),

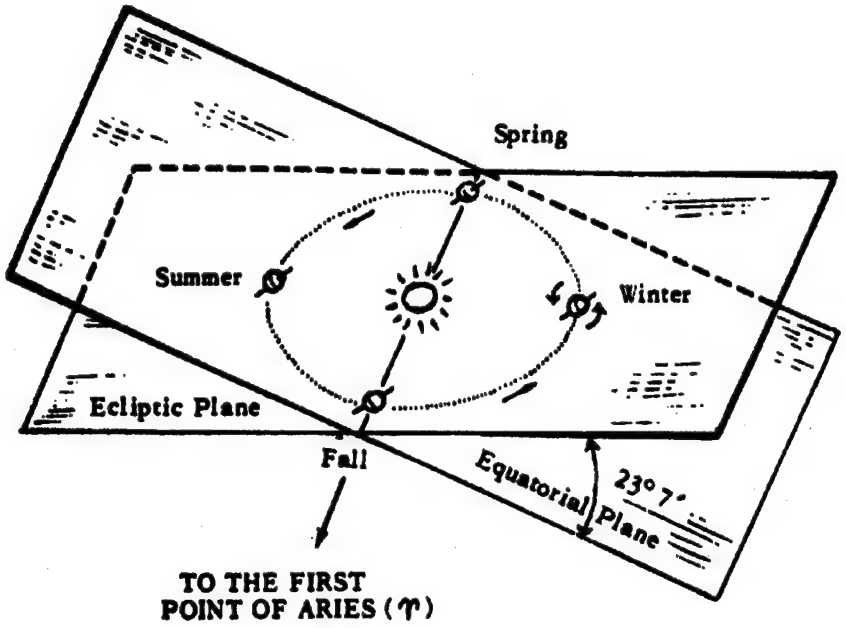

Figure 1. The vernal equinox or first point of Aries. [9]

and **argument of perigee**(ω) (orientation of vector pointing to lowest altitude of orbit). The position of the satellite is determined by **epoch time**, which is the time the satellite passes either perigee or crosses the equatorial plane (this crossing point is known as ascending node) (Figure 2).

The other method of determining an orbit is by location and velocity vectors. The position is denoted by (x, y, z), and the velocity is denoted by the change in position, or ($\dot{x}$, $\dot{y}$, $\dot{z}$).

At lower altitudes, a satellite moves faster. An orbit is considered low-earth if the period is less than 360 minutes. The simulated satellite images used for testing the FST NN and HMM classifiers in chapter 4 are from a low-earth orbit with a period of 90 minutes. Another characteristic of this orbit is that it is sun-synchronous. Sun-synchronous means that the orbit plane remains aligned with respect to the sun. This is a phenomenon that occurs in some high inclination orbits, and it provides the benefit of the satellite passing over a point on the earth

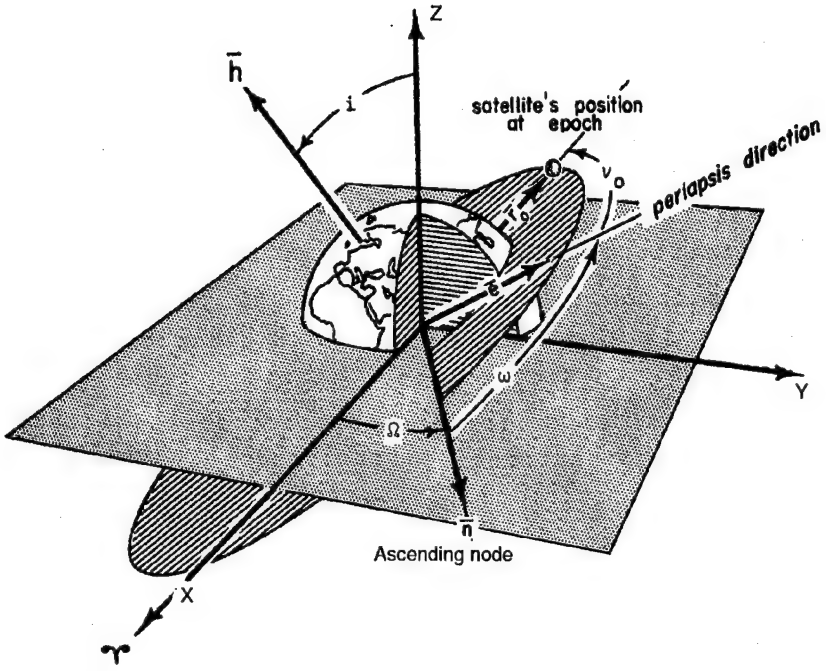

Figure 2. Some classical orbital elements: Ω - right ascension, ω - argument of perigee, i - inclination. [1]

at the same time of day throughout the year. This occurs because the orbit plane is perturbed approximately one degree per day (Figure 3).

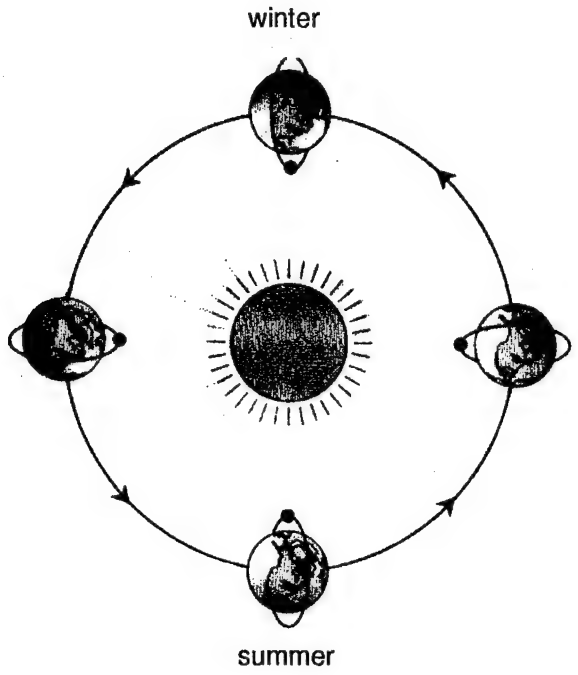

Figure 3. A Sun-synchronous orbit keeps the same orientation to the sun throughout the year. [9]

The attitude of a satellite is also important. The attitude is normally described in terms that are used for aircraft: roll, pitch, and yaw. For this research, a normal satellite will be nadir pointing and 3 axis stable. Any roll, pitch, or yaw will be considered anomalous.

2.2 Traditional Pattern Recognition Approaches

One of the most basic approaches to pattern recognition is the statistical classifier. R. A. Fisher [7] in 1936 was one of the first scientists to develop a statistical classification method and apply it to a real problem. After the computer was invented, the uses for and types of statistical classifiers have exploded. R. O. Duda [3] provided a thorough treatment of statistical classification, with bibliographic references, in 1973.

Non-statistical classification methods include the k-nearest neighbor and the neural network. The neural network approach to pattern recognition was developed in the late 1950s beginning with Rosenblatt's perceptron [17]. Since then, a large variety of techniques have been developed for structuring and training neural networks.

Traditional pattern recognition approaches have been successful on many image recognition problems, but the problem of recognizing images of 3D objects viewed from various angles or illumination conditions has required some new approaches.

2.3 Feature Space Trajectory (FST) Neural Network (NN)

Neiberg and Casasent [12] developed a neural network that classifies images regardless of 3D aspect distortions. It is very similar to a nearest neighbor classifier but it has the ability to interpolate between training samples. For the experiments performed by Neiberg and Casasent, it is shown to perform better than several other neural network classifiers and to accurately reject non-object data [13]. This FSTNN is able to account for 3D aspect distortions by describing each class as a trajectory in feature space.

2.3.1 FST NN Performance. Neiberg and Casasent tested the FST NN against a simulated infrared military vehicle database (TRIM-2) which contains 126 objects. The vehicles were present in scenes and had to be detected and enhanced prior to feature extraction. The database contained 21 aspect views of six different vehicles. Eleven images per class, a total of 66, were used for training and the remaining 60 were used for testing.

The feature space for their experiment was wedge-sampled magnitude-squared Fourier features. The paper addressed whether to use the dc value, whether the features should be normalized (so the total energy sums to 1.0 for each image), and whether the pixels surrounding the dc value should be suppressed. Various combinations were tested and the best results were achieved without normalization and without suppressing the values surrounding the dc value. Comparison experiments with other classifiers were performed using these features. Fourier rings were also tested with very good results. For both wedges and rings, 16 features per

image performed better than 32. More features, normalizing, or dc suppression requires a larger training set to achieve the same results [12].

The other classifiers used by Neiberg and Casasent for comparison were the three-layer feedforward Backpropagation NN (BPNN), piecewise quadratic NN (PQNN), the direct storage nearest neighbor (DSNN), and the condensed nearest neighbor (CNN) classifiers. The FST NN was the only classifier to achieve 100% on the test set with the BPNN coming in second at 93.3%. One of the primary advantages of the FST NN is that it generalizes better than other classifiers and it doesn't have a problem with memorization. Also, it doesn't need a large training set to ensure good generalization. Additionally, iterations are not required for the FST NN unless the middle layer is pruned, while the BPNN requires more than 60,000 iterations. The BPNN performed best when the middle layer nodes were less than 15 (or roughly 3 per class). The on-line computational load for the FST NN was about 34% higher than the BPNN.

Another capability of the FST NN is pose or aspect view estimation. This is accomplished by interpolating along the trajectory between the two nearest middle layer nodes. Each middle layer node represents a training image with a known aspect view. A limitation of the FST NN as applied by Neiberg and Casasent is that aspect estimation is limited to one degree of freedom. Only azimuth was considered, and this could be a problem with some applications. For the SOI problem two degrees of freedom are required.

In summary, Neiberg and Casasent have developed a technique that performed well on the TRIM-2 database, acheiving 100% accuracy on some tests. This classifier appears to have great potential in a variety of classification problems such as 3D aspect distortions, aspect estimation, and motion characterization.

2.3.2 FST NN Description. The FST NN is very much like a nearest neighbor classifier. The difference is that a nearest neighbor classifier uses the distance to the nearest feature vector in the training database, but the FST NN classifier uses the distance to the

nearest trajectory (Figure 4). A segment of the trajectory represents the transition between aspects represented in the training database.

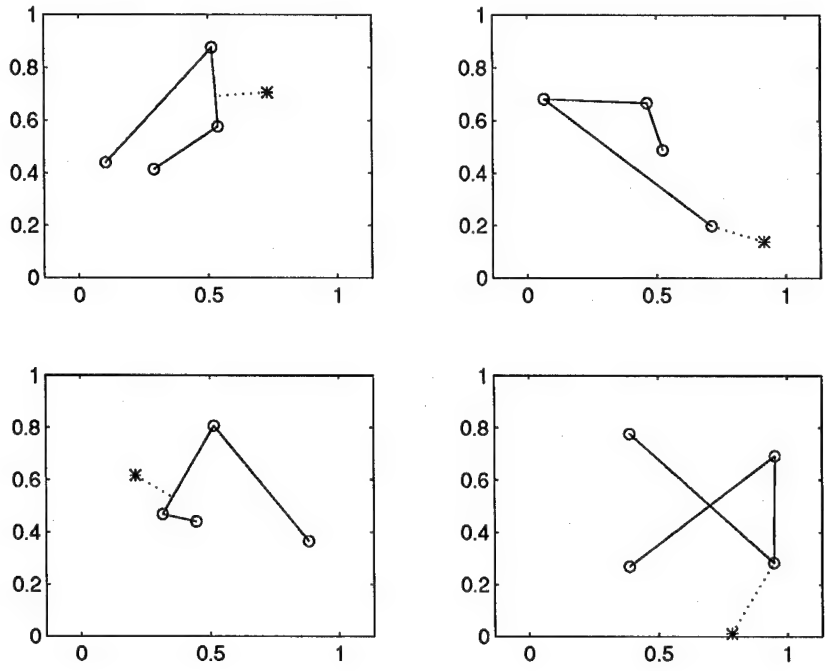

Figure 4. Examples of distance from point to trajectory in hypothetical 2D feature space. Distance shown by dotted line.

Neiberg and Casasent used the FST NN to classify images of military vehicles, using various aspect views of each vehicle as the training set. The closest trajectory determines the class of the test vehicle, and the position on the trajectory determines the pose (Figure 5).

The satellite anomaly problem requires a slightly different approach for a couple of reasons. Since the aspect view of a satellite can vary over a solid angle of 4π, many more training trajectories are required. Also, a single view of the test object is not adequate, since motion and the change in aspect over time must be considered. Additionally, training data from the anomalous class is not typically available, so the algorithm must be designed to detect sequences from the normal class and all others are classified anomalous.

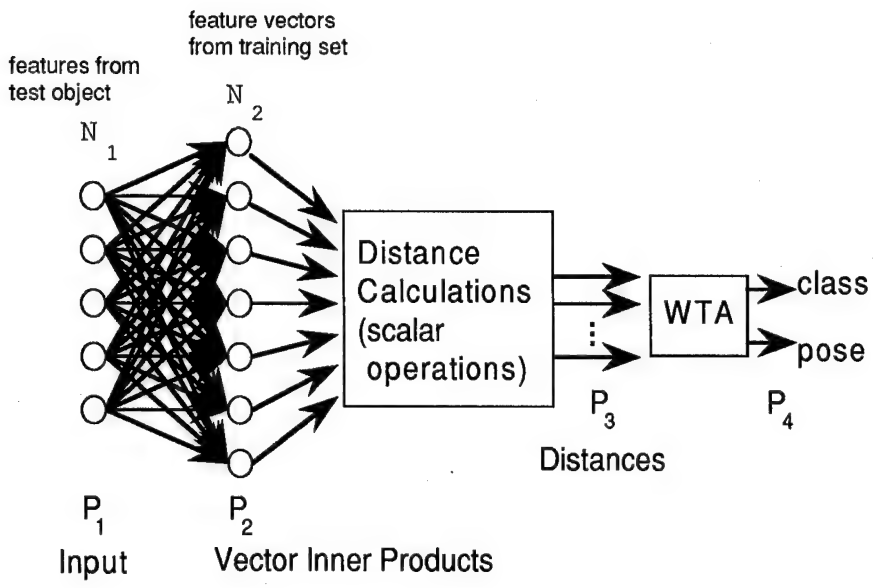

Figure 5. FST NN architecture. [13]

2.4 *Hidden Markov Models*

The Hidden Markov Model (HMM) is a statistical classification technique that has been used for speech recognition since about 1975 [11]. Recently, Fielding and Ruck [6] [5] demonstrated the use of the HMM as a spatio-temporal image sequence classifier. This approach uses a sequence of images rather than a single look to improve classification.

2.4.1 HMM Performance. The HMM technique used by Fielding and Ruck successfully classified moving light display images and simulated image sequences of military vehicles. They used Fourier features from the 7 x 4 rectangle of low frequency, magnitude only, coefficients. This produced a 28 D vector. Each feature vector component was statistically normalized. The features from the training set were vector quantized into clusters that represent various aspect angles. A sequence of images was transformed into a sequence of feature vectors that would produce a trajectory on the aspect graph. The HMM algorithms determine which class of vehicle most likely produced the trajectory.

A five class problem was considered and classification accuracies as high as 100% were obtained. Tests were also performed to classify type of object and direction of motion. The

HMM classifier performed well even when noise was added and it consistently outperformed the single look nearest neighbor classifier. Also, a real image sequence was successfully classified using an HMM trained on synthetic data. Fielding and Ruck showed that an HMM uses temporal information and will produce superior classification results when compared to single frame techniques.

2.4.2 HMM Description. Hidden Markov Model classifiers are not easy to understand, but a good place to start is with a simple Markov process. Let us consider a simple three state Markov process. The topic is car color and the three states for this example are red, white, and blue. This is a **discrete** Markov process, since a car (for our purposes) is only red, white, or blue, and not shades in between. If it were a **continuous** model, the cars could be any shade of color, and the car's state would be the color that is the closest match. The transition probabilities represent the chance that the car will go from one state to another. A transition occurs when the car is painted. The model shown in the figure is called **ergodic** because any state can be revisited (Figure 6).

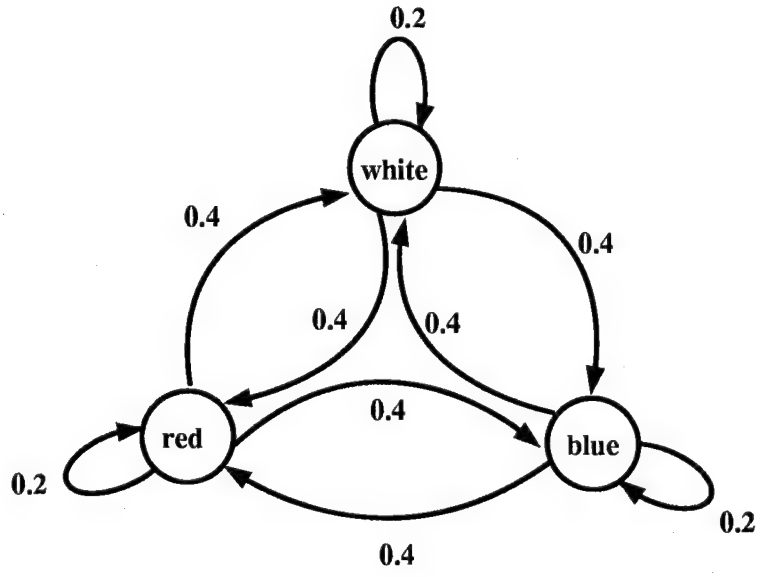

Figure 6. Example of an Ergodic hidden Markov model.

In the example shown in figure 6 there is a 0.4 probability that a white car will be painted red. For two transitions, the probabilities are multiplied. The probability that a white car will be painted white then later be painted red would be 0.08.

Another type of model is the **left-right** Markov process. For this example, let the state be the side of the car being viewed. Let's just consider three states: front, side, and back. This process models the view of a car driving by in a normal manner. First you would see the front, then the side, then the back. If a person records which side of the car is viewed at one second intervals, the transition probabilities may look something like this (Figure 7).

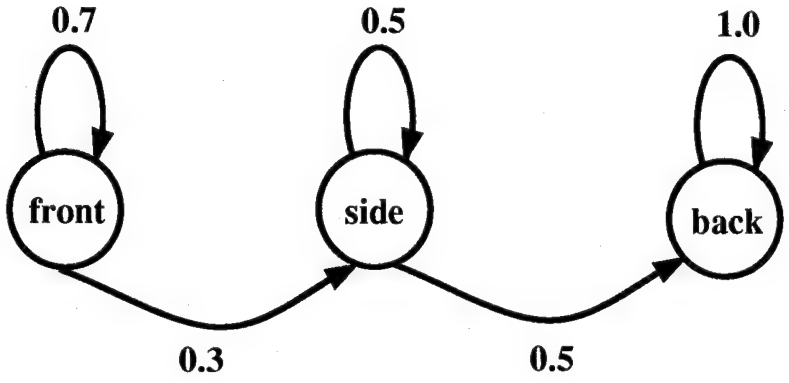

Figure 7. Example of a left-right hidden Markov model.

In this model, once a state is left, it will never be revisited. A possible observation sequence for this model may be: {front, front, front, front, front, side, side, side, back, back, back, back}. From this model, there is a probability of 0.009 that this sequence would be observed. Now, if a car was driving in reverse, a possible observation sequence may be: {back, back, back, side, side, front, front, front, front, front, front}. From this model, the probability of this sequence being observed is zero because right to left transitions cannot occur. If a car were crashing, the observation sequence may be: {top, side, back, bottom}; and the probability of this observation would also be zero because the model only accurately represents a normal drive by. The key to the HMM classifier is to find a model that will maximize the probability of an observation sequence for a given class, and observation sequences that are not from the class should have a much lower probability.

The Hidden part of the HMM comes from the fact that you may not be able to directly observe the state. If one uses an infrared sensor to detect the car, the front, side and rear aspects of the car may have different readings. A given reading may not relate directly to a given aspect, but probability density functions could be derived to represent the various aspects. One might guess that a three state model would work best, but a four or five state model may perform better because the best states may not relate to front, side, and back aspects. Also, one would expect a left-right model to work best, but if one of the states relates to front and back corner views, an ergodic model would work better because this state would need to be revisited.

There are algorithms to help one find the best Markov model for the problem. Starting with a sequence of observations, one would use the Viterbi algorithm to find the best values for each state. Next, one would use the Baum-Welch reestimation algorithm to find the transition probabilities. These transition values would be selected to maximize the probability that the observed sequences came from the model. Once the model is determined using Viterbi and Baum-Welch, the forward or backward algorithm is used to find the probability of a given observation sequence given the model. These algorithms are spelled out in Rabiner and Juang [15], Poritz [14], and Rabiner [16]. Also, the commercial software package by Entropic Research Laboratory called Hidden Markov Model Toolkit (HTK) has these algorithms [4].

For the satellite anomaly detection problem, the observation sequence is the sequence of feature vectors that are derived from an image sequence, and the states are also vectors in this feature space. The concept of states for this problem is not as clear as the car example above, because one state may represent numerous aspect views.

III. Methodology

This chapter covers the methodology, beginning with the production of simulated data for the experiments, how it differs from real data, and some preprocessing that may be helpful on real data. The second part of this chapter covers the specific classifier algorithms and features tested for this thesis.

3.1 Simulated Images

3.1.1 SatTools. This satellite anomaly detection research is based on experiments using simulated satellite images. These images were produced using a program called SatTools. SatTools allows the user to choose an orbit, a satellite model, an optical sensor, and other parameters to produce a sequence of images that simulates real optical space surveillance data. The SatTools software has three parts: **stem**, to produce the orbit; **satsig**, to perform ray tracing, and **satsim**, to convert the data into an image.

With the stem input file, the user specifies the orbit using classical orbital elements or position and velocity vectors. The user also specifies the attitude of the satellite, the start/stop times and time step between image frames. The output file describes the motion of the satellite, providing the user with azimuth, elevation and range from the viewer to the satellite, the angles from the satellite to the viewer, and the angles from the satellite to the Sun for each frame. Also, latitude, longitude, and altitude are given for the satellite at each frame in addition to some other information not used in this thesis.

Satsig uses the stem motion file to determine the aspect of the satellite being viewed, then performs ray tracing against the satellite model. The user specifies the number of ray bounces, whether light is reflecting off the Earth (earthshine), and whether the Earth's shadow is considered.

Next, satsim takes the satsig output and transforms it into an image. The user can specify the optical characteristics, including wavelength, detector size, sample time and aperture size. The user can also specify atmospheric effects, noise, and jitter.

3.1.2 Simulated Image Production. Simulated image production takes a significant amount of time. The SatTools program runs on Silicon Graphics workstations. To make simulated daytime images, the earthshine setting is activated. This increases the runtime for a sequence of 20 images from less than one hour to 10-20 hours, and 175 of these sequences were produced.

Since the exact characteristics of the AMOS optical sensor are not available at this classification level, the SatTools data was produced assuming ideal conditions using visible wavelength, no atmospheric turbulence, no noise, and no jitter. Two batches of image sequences were produced. The first batch (data set 1) covers a time frame of 17 days . The images were produced using an orbit with a semi-major axis of 6771 km (which converts to an altitude of about 400 km or a period of 92 minutes), inclination of 101 degrees, and eccentricity of 0.0004 (nearly circular, so argument of perigee is not important). The epoch time and right ascension were set so the satellite would have a descending pass (north to south) over the site between 0200Z and 0400Z and an ascending pass (south to north) between 1400Z and 1600Z. This corresponds to an early morning ascending pass, and a late evening descending pass. All passes were imaged as long as they were at least five degrees above the horizon. Each sequence is made up of twenty images, created at 10 second intervals. This 200 second window was centered on the highest elevation angle of the pass. For the training images, the right ascension was not adjusted over the 17 day time frame, so the time of day was about one hour off by the 17th day. For the purposes of this experiment, this is close enough to sun-synchronous. After 17 days, with an average of three passes per day, the data covers a broad range of possible viewing angles; and the orbit trace begins to repeat, so training data production is stopped. This training set includes 50 image sequences (Figure 8).

The first test set included 20 normal sequences and 30 anomalous sequences. The normal sequences were produced by using the same orbit parameters of the training set, but perturbing the right ascension so the images would fall in a gap. If the sequence was not forced to fall in a gap, the data would be very close to the training set and classification would be too easy. The anomalous test set was produced with the same orbital parameters as the training

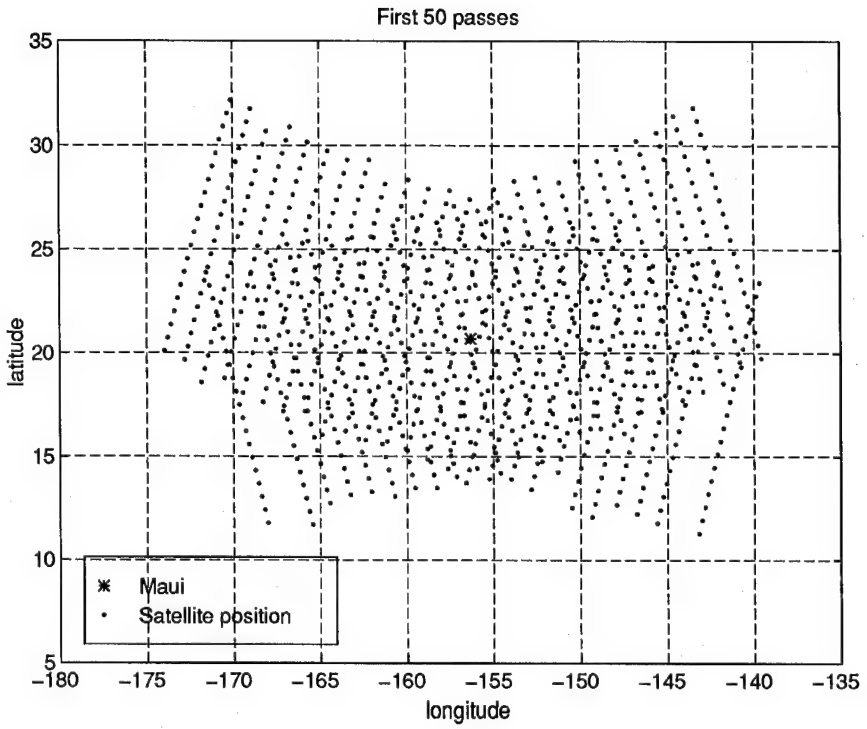

Figure 8. First set of training data.

set but with various amounts of rotation in the roll, pitch, and yaw axis. Sequence examples are shown in figure 9 showing a couple of normal passes and an anomalous pass. Not all anomalous passes were this obvious. The rotations were added at various rates between 0.2 and 3.0 degrees per second. These were added one axis at a time, and in some passes two axes rotation was simulated.

The second data set was produced after learning from experiments on the first data set. The orbit parameters were basically the same except the right ascension was perturbed about 0.9 degrees per day to keep the orbit truly sun-synchronous. Only descending passes were considered, and the pass elevation was limited to at least 10 degrees above the horizon. Also, the time interval between images was reduced to 8 seconds. These changes result in a closer sample spacing (Figure 10) and should lead to better classification results. The normal test set was produced in a similar manner to the previous normal test data set, by forcing the orbit to fall in gaps between training passes. The anomalous test set was produced by introducing roll, pitch, and yaw based in a semi-random manner. This was to reduce chance alignments that were evident in the first data set because of one axis rotation.

For the final test, shot noise is added to the second data set. Shot noise is the natural phenomena caused by random arrival rate of photons on a sensor. This noise is simulated using a Poisson rate scaled by the intensity of the original image (Figure 11). The calculation requires normalizing the pixel values in the image matrix F, then scaling by the $\bar{K}$ value.

$$Fnorm = F/\Sigma F$$

$$Fscaled = Fnorm * \bar{K}$$

Next, each element of the image matrix $Fnew$ is determined based on a poisson distribution with a rate of $Fscaled$.

$$Fnew = poisson(Fscaled)$$

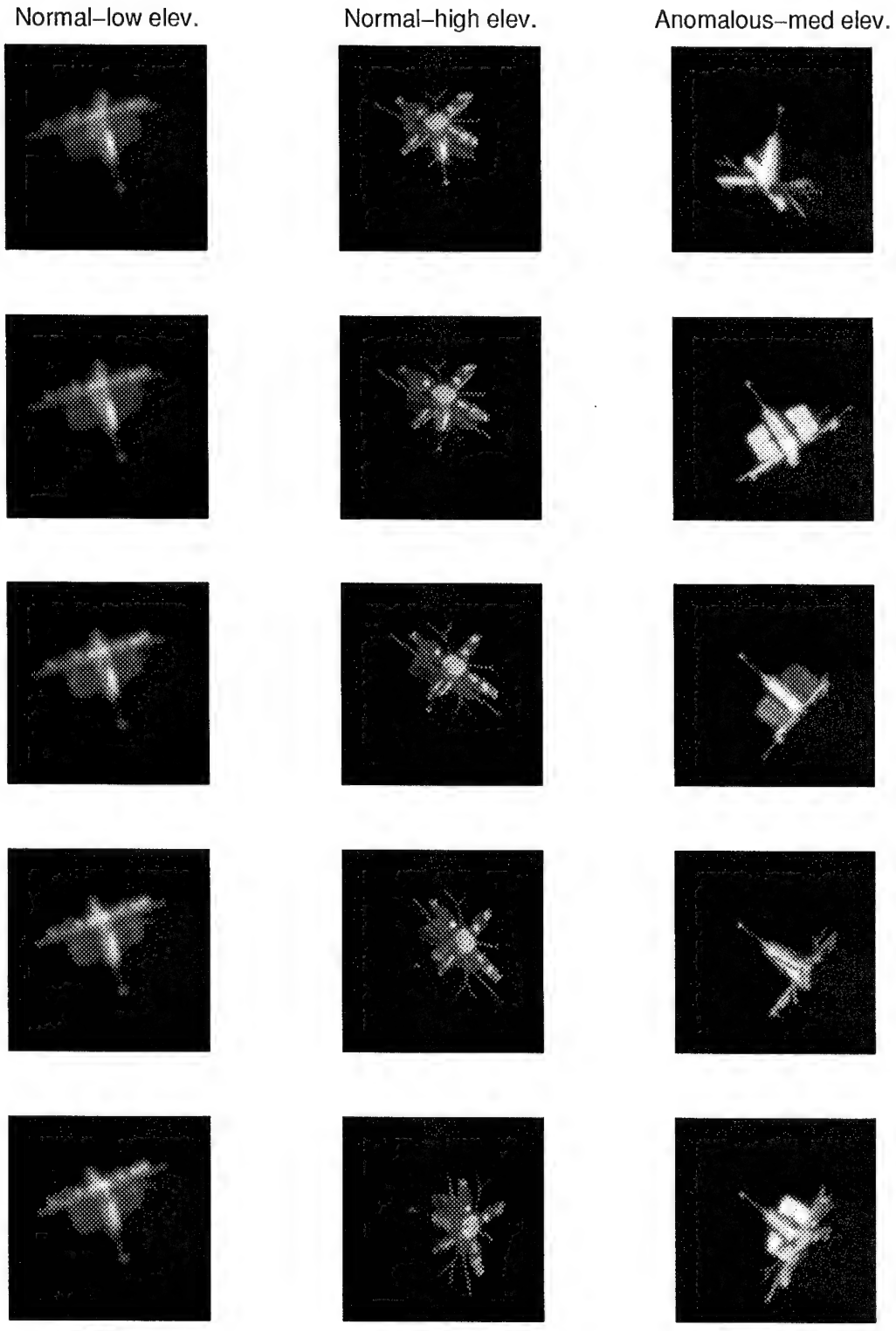

Figure 9. Samples of normal and anomalous image sequences.

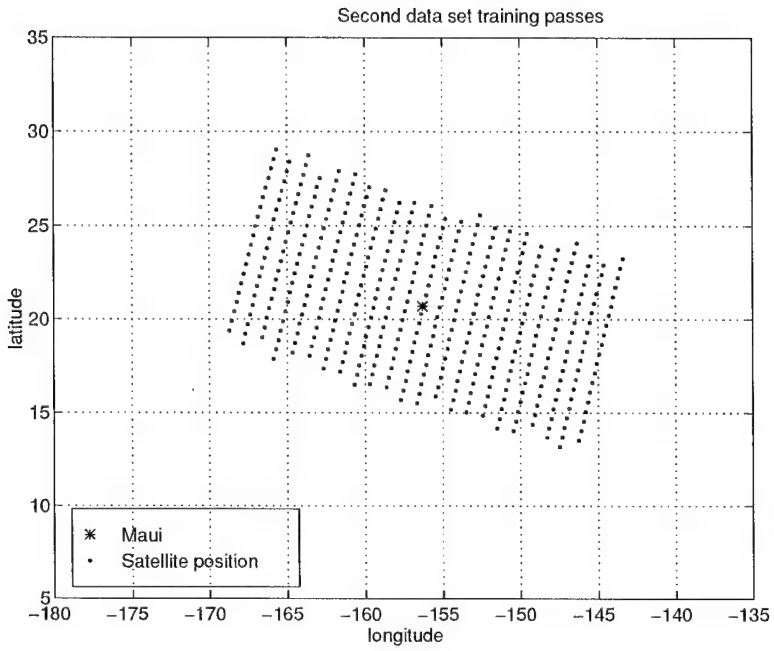

Figure 10. Second set of training data.

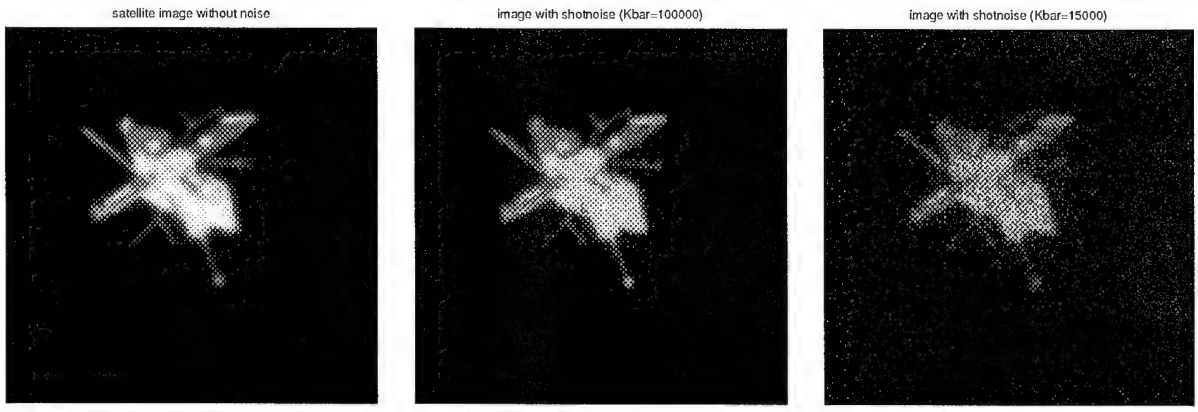

Figure 11. Example of an image with various levels of shot noise.

3.2 Real Images

Before SatTools was available, the real images were examined and studied to find what problems may exist that would prevent accurate classification. Some possible problems and suggested solutions are covered in this section.

The images provided by Air Force Maui Optical Station (AMOS) have a wide range of brightness variation. The gray scale intensity of some of the images is so low that the space objects were visually undetectable. Also, the noise in some images was higher than the object pixels in other images. I made the assumption that a classifier would perform better if the intensities could be adjusted to 'normalize' intensity levels. My goal in the pre-processing was to produce an image that visually minimized noise or extraneous bright areas while bringing the object to a level that could be easily detected.

3.2.1 Thresholding. The brightness threshold is the approximate minimum pixel intensity that defines the object. Pixels below the threshold are declared background, that for these images should be black. A procedure was needed to pick the best threshold for each image regardless of the quality or original brightness level of the original image. Methods used by other researchers include mean or mean plus standard deviation [8]. By trial and error, I found the best threshold for the AMOS data to be mean plus .07 times the variance (Figure 12). For producing binary images, pixels below the threshold were given a value of zero and pixels above the threshold were given a value of one.

3.2.2 Hyperbolic Tangent Warping. In order to keep gray levels, another method of applying the threshold was developed using the hyperbolic tangent or tanh function. This maps the original pixel values (0 to 255) in a non-linear operation to values between -1 and 1, with the threshold mapping to 0. Then the values are rescaled to fit the original pixel value range of 0 to 255. (Figure 13). The equations for the tanh transformation follow:

$$t = \mu + 0.07\sigma^2$$

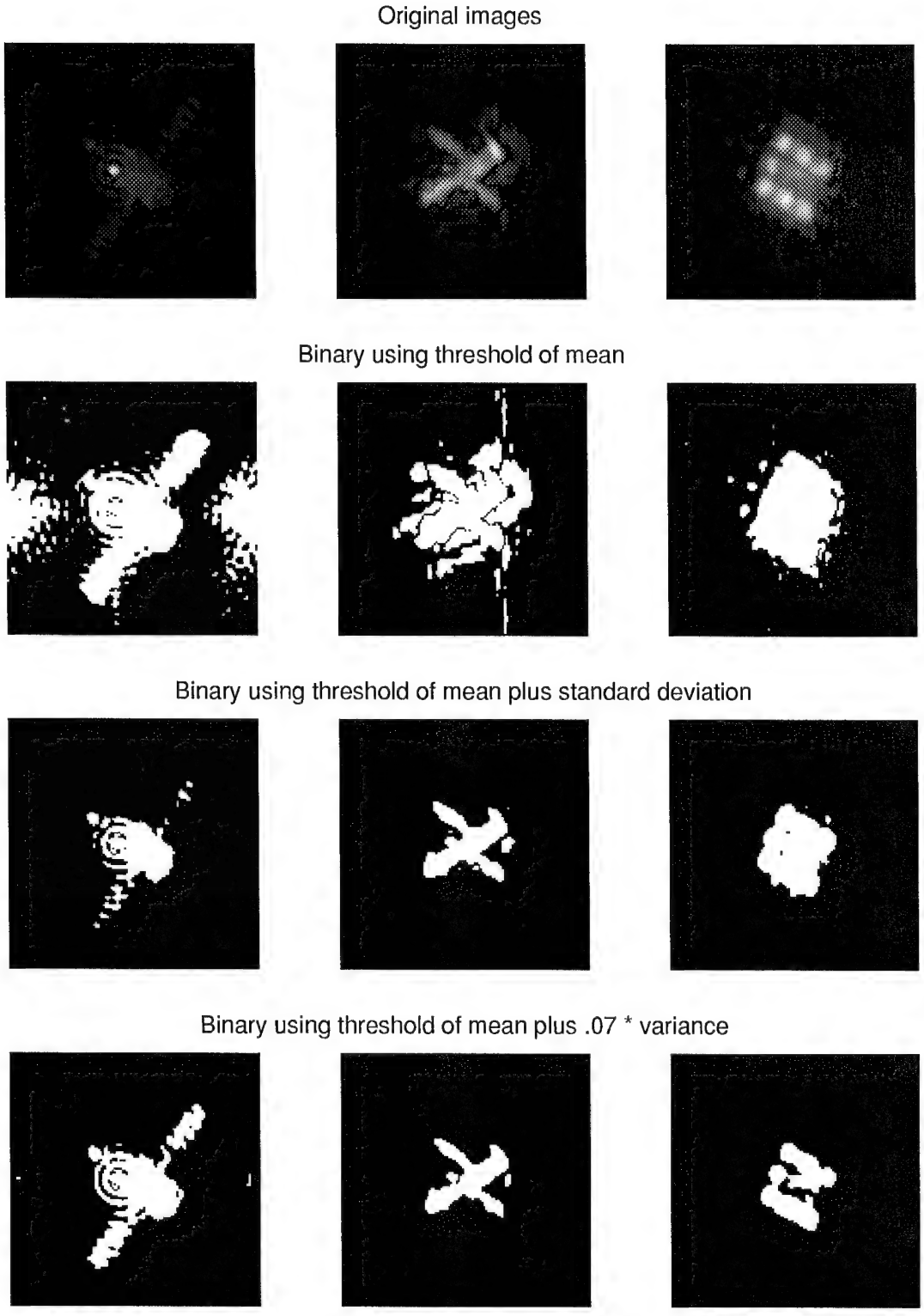

Figure 12. Examples of various thresholds.

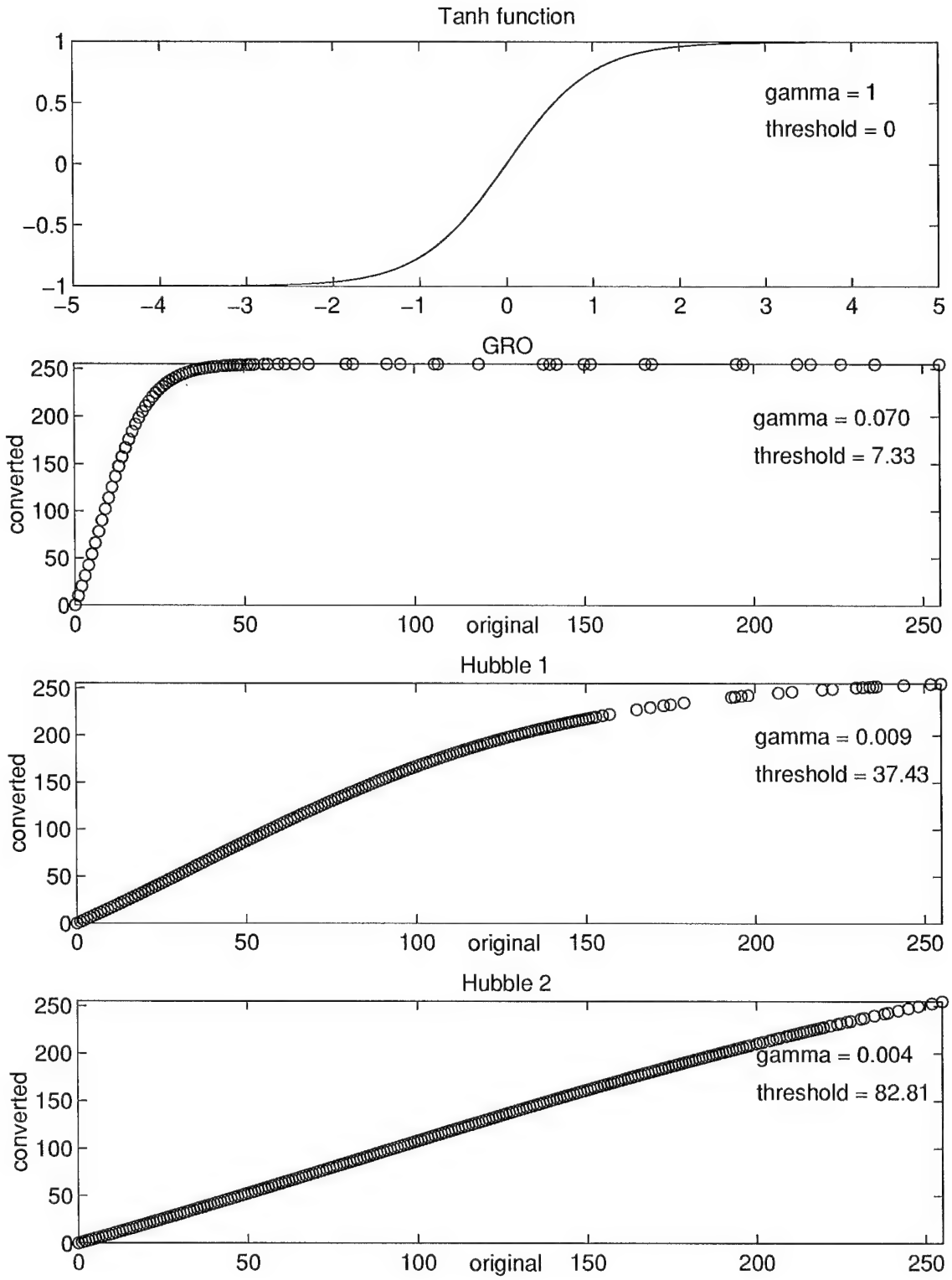

Figure 13. Examples of pixel values transformed by tanh function.

$$\gamma = (2/\sigma)^2$$

$$temp = (tanh(\gamma(F - t)) + 1)$$

$$temp2 = temp - min(temp)$$

$$newF = temp2 \times 255/max(temp2)$$

Where t is the threshold, μ is the mean value of the pixels in the image matrix F, σ is the standard deviation of the pixels in F, and γ controls the slope of the tanh function. An adaptive γ based on a fraction of the variance was found to work best. The effect of this tanh processing is histogram reshaping. It was compared to the 'histeq' command in Matlab (Figure 14). Although this is a subjective assessment, the tanh process appears to work better. Dim images, like GRO, are brightened better with tanh than with histeq; but good images, like Hubble1, are nearly unchanged by tanh while histeq over brightens the image, bringing out extra noise.

3.2.3 Filtering. After each image was thresholded, other methods of processing were considered for 'cleaning up' the image. One that works well is median filtering. A 3 by 3 median filter smooths out irregularities in the image and fills in small gaps without significantly changing the shape of the object (Figure 15); however, it is computation intensive.

3.3 Features

An important step in implementing any pattern recognition algorithm is to find features that properly represent the data for classification. This section examines various magnitude only Fourier features, beginning with a basic description of the candidates. The latter half of this section covers the pros and cons of each type of feature considered.

3.3.1 Why Fourier Features? Magnitude only Fourier features are good for image pattern recognition because they represent the shape of an object but are shift invariant. Also, they can be a powerful data reduction tool. For the larger components, or overall shape

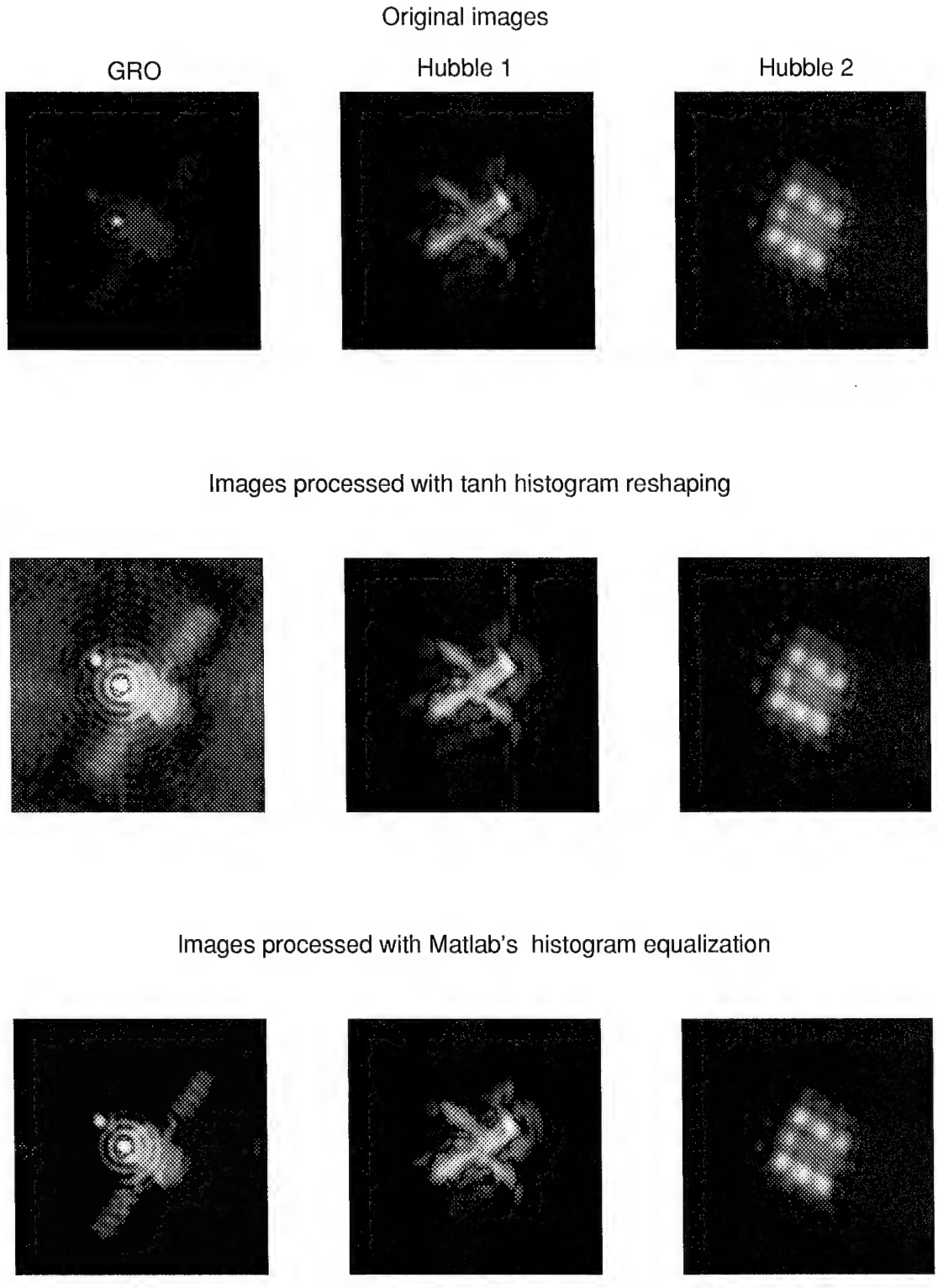

Figure 14. Comparison of using tanh versus Matlab's histogram equalization.

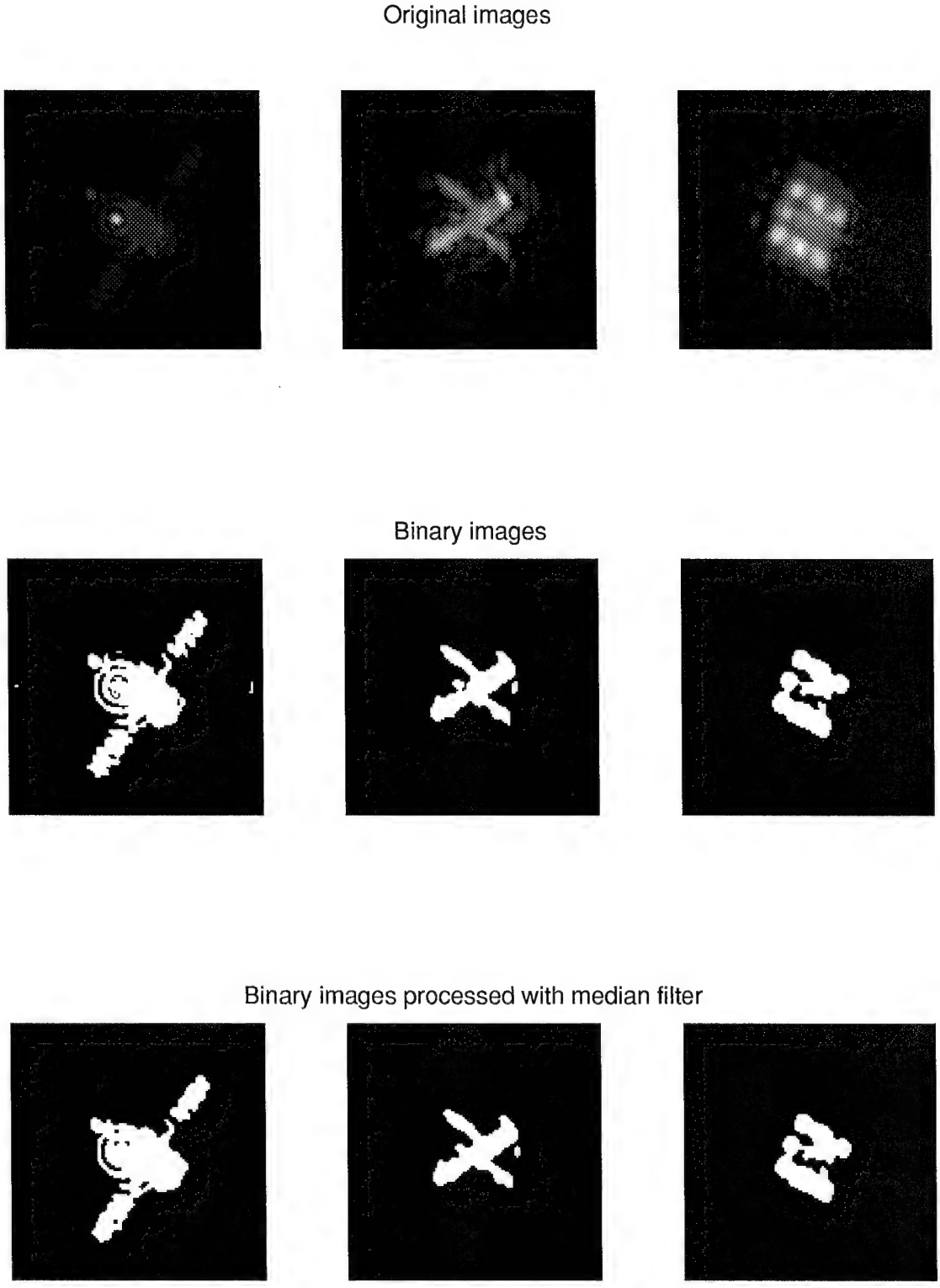

Figure 15. Comparison of binary image with and without median filtering.

of an object, the low frequency coefficients are known to perform well. For representing orientations or alignments, wedge sampled features are known to work well. The next two sections describe the types of Fourier features tested.

3.3.2 Wedge vs Block. The wedge Fourier feature space is a good candidate for the satellite anomaly detection problem because it is shift, scale and brightness invariant (if normalized), while still being sensitive to rotational changes. Wedge Fourier features are obtained by first taking the 2D Discrete Fourier transform (DFT) of the image, then dividing the space into wedge shaped regions. All the values within each wedge are summed and divided by the number of pixels in the wedge (Figure 16). Only half of the wedges are used

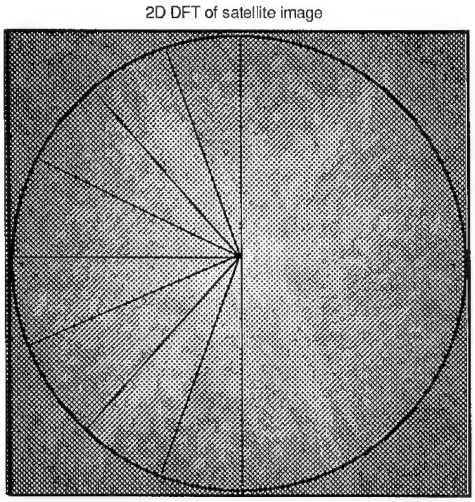

Figure 16. The eight wedge DFT boundaries.

because the 2D DFT is symmetric. The value obtained from each wedge becomes an element of the feature vector. Using many small wedges creates a very rotationally sensitive feature space, while using few larger wedges creates less sensitive features. Neiberg and Casasent [12] used 16 wedges very successfully on infrared images of military vehicles. For this thesis, both 8 and 16 wedge features are tested.

Another proven Fourier feature set is the 2D DFT low frequency coefficients. Fielding and Ruck [6] used a 7 by 4 block of coefficients, for a 28 D feature vector. For this thesis, however, only 18 coefficients are used (Figure 17). This is to keep the feature vector approximately the same size as the 16 wedge features.

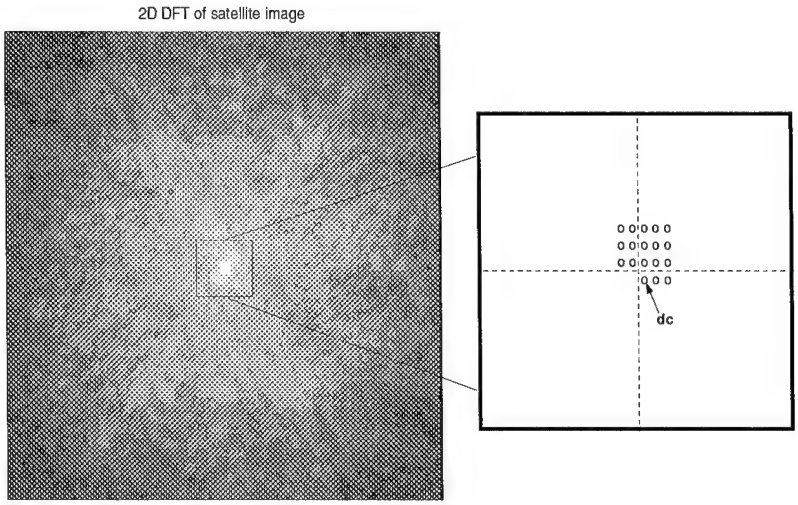

Figure 17. The block of 18 DFT coefficients.

3.3.3 Normalization. Two types of normalization are applied to the features: statistical and energy. Statistical normalization is a technique to keep the scale of the features approximately equal. This is important when the mean and variance of the elements of a feature vector are not of the same magnitude, especially when the distance measure is Euclidean. Without statistical normalization, the feature elements with the largest variance effect the outcome more than the elements with a small variance. The feature elements with a small variance may even become useless. With block Fourier features, statistical normalization is important because the coefficients closer to dc are much larger than the values farther out.

With wedge Fourier features statistical normalization is not required because each element of the feature vector has approximately the same mean and variance; however, energy normalization could be effective. Energy normalization is achieved by dividing each value of

a feature vector by the square root of the sum of the squares. The result is a feature vector that is invariant to brightness. If DFT wedge features are energy normalized, they will be scale invariant. This is not important for the data generated for this thesis, but it may be important for real data if the images are not collected with a consistent field of view.

3.4 FST NN

The FST NN algorithm from Neiberg and Casasent [12] had to be tailored to work on the satellite anomaly problem. Instead one trajectory per class, there must be a set of trajectories to represent the class, since the satellite can be viewed from many angles over the entire aspect sphere, not just an aspect circle. Secondly, the test input must be not just one image, but a sequence of images, so anomalous motion can be detected. Thirdly, instead of just using feature space distance, there must be a measure for how well the test images follow a normal sequence. The modified FST NN architecture is shown in figure 18.

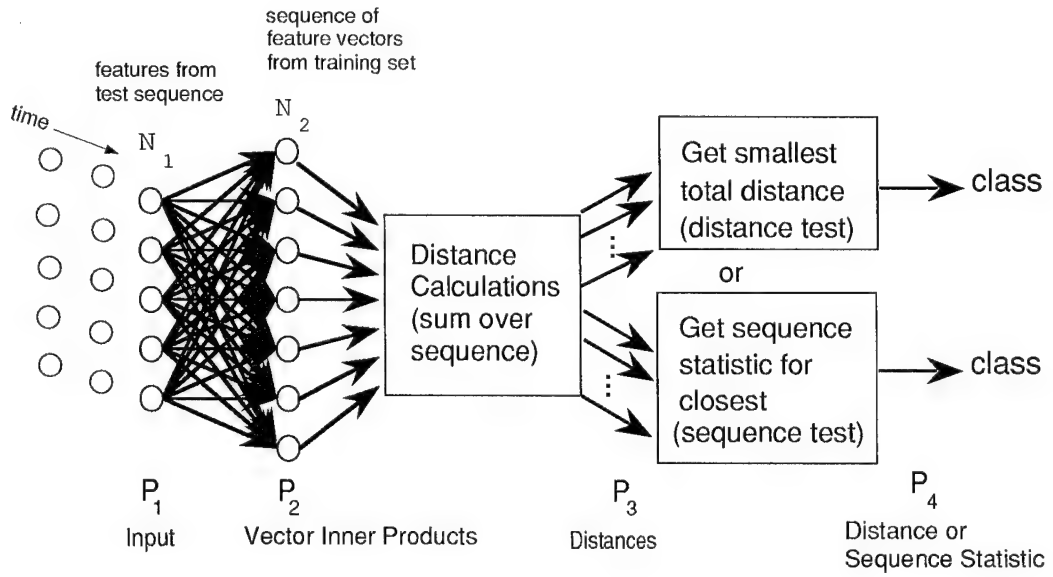

Figure 18. The modified FST NN architecture.

The FST NN algorithm was tested against the two sets of data. The first data set was used to test many different approaches. And the second data set was tested with the most promising combinations.

3.4.1 The Distance Approach. The FST NN was applied using the summed distance from the test sequence to the nearest normal sequence is used for classification. Appendix B shows the Matlab [18] code for the algorithm. If the distance is below a certain threshold, the object is classified as normal, and if the distance is exceeded, the object is classified as anomalous. Anomalous training data is not used because for real applications, anomalous data may be scarce. A problem exists about where to put the threshold since the training set does not include sequences from the anomalous class. A threshold can be derived statistically from the leave-one-out data, or by creating simulated anomalous statistics. For this experiment, I borrowed anomalous data from the other data set (a different orbit), to approximate using anomalous data from different satellites.

3.4.2 The Sequence Approach. Another approach was tested which measures how well the test trajectory matches the sequence of the nearest training trajectory. This is accomplished by first selecting the best training trajectory based on the distance approach then checking how well the test trajectory follows the sequence. If the sequence matches within a certain limit, then the class is normal. A simple sequence test was developed that works well. The actual sequence is subtracted from an ideal sequence, then the standard deviation of the result is used for classification. If this value is below a certain threshold the sequence is classified as normal. If the value is above this threshold, the class is considered anomalous.

Another sequence approach was tested where the feature space distance was ignored and the lowest sequence test value was used, but this idea was abandoned because of poor results.

Both the distance approach and sequence approach were tested with the following six feature types:

Feature type	DC	Normalization	Vector length
16 wedge Fourier features	yes	Energy	17
8 wedge Fourier features	yes	Energy	9
8 wedge Fourier features	no	Energy	8
Block Fourier features	yes	None	18
Block Fourier features	yes	Statistical	18
Block Fourier features	yes	Energy	18

Non-normalized wedge features were initially considered but abandoned because of poor results.

3.5 HMM

The HMM algorithm can be implemented many different ways. Any number of states or mixtures per state can be used, only limited by the number of free parameters. This research tests 1, 2, 5, and 10 state ergodic models, and 2, 5, 10 and 15 state left-right models. Continuous observation distributions are used with both one and four Gaussian mixtures. For the first data set, the best wedge features and block features from the FST NN experiment are tested using all sixteen HMMs. For the second data set, only the two best ergodic and two best left-right models are tested using both wedge and block features.

The HTK software [4] was used for this experiment. This requires the user to build a prototype file of initial states and transition probabilities. The initialization algorithm clusters the training data using k-means, then finds the best states and transition probabilities. The Viterbi algorithm then finds the best states for the test data. The Baum-Welch algorithm reestimates the model parameters. For the left-right model, the user also had to choose whether a state can be skipped. Two different starting conditions are tested (appendix - sample prototype files for the HMM).

3.6 Classification Threshold

After the FST NN or HMM is trained, a classification threshold must be determined. An ideal threshold would be found by testing much data from each class and selecting a threshold that minimizes the number of missed classifications. This is impractical for space object identification because anomalous data will probably not be available for training. Therefore, the threshold must be derived from the normal data in the training set or borrowed from a set where anomalous data exists.

Four methods are considered: (1) A threshold can be derived statistically from the training data by **leave-one-out** tests. This approach is not perfect because there is a gap left in the training set from the sequence being tested, and therefore the mean would be a better basis for a threshold than the maximum. (2) One method of creating anomalous data is to borrow anomalous data from another data set (a different orbit or different satellite). This is called the **borrowed anomalous** method. (3) Another way to create anomalous data is to mix the data from the training set (example, take the first image from the first sequence, the second from the second sequence, etc). This is called the **mix-up** method. (4) For the FST NN sequence test, anomalous statistics can be created by forcing the test sequence to match up with a training sequence that is not close in feature space. This is the **forced mismatch** method. For the anomalous methods (methods 2, 3, and 4), the threshold is based on the minimum value. Any of these thresholds can be adjusted up or down depending on whether one wants to increase the probability of false alarms or missed anomalies.

IV. Results

This chapter presents the results of the FST NN and HMM satellite anomaly detection tests with a full analysis. Each of the first four major sections covers the results for specific data sets, and the subsections cover the results by algorithm. All results are displayed in a similar style so comparisons are easily made. Two types of graphs are used: one shows the distribution of actual result values for each class; the other is a receiver operating characteristic (ROC) graph, showing probability of false alarms (P_{FA}) versus the probability of anomaly detection (P_D). The ROC graphs and distributions show the type of results that are possible. Actual classification results depend on the selection of an appropriate threshold. The fifth section of this chapter covers the selection of a classification threshold and the last section summarizes the results.

4.1 Anomaly Detection Tests on the First Data Set

The first data set as described on page 17 includes 50 training sequences and 50 test sequences. Of the 50 test sequences, 20 are normal and 30 are anomalous.

4.1.1 FST NN. The FST NN anomaly detection tests were successful. Of the combinations tested, the best results were achieved by the non-normalized block features using the sequence test. (Figures 19 and 20).

Comparing block Fourier features to wedge Fourier features, there does not appear to be an advantage to either feature set. Eight wedge features outperformed 16 wedge features and adding a dc feature did not improve performance. Energy normalized wedge features performed better than non-normalized or statistically normalized wedge features. This is good since energy normalized wedge features have the benefit of being scale and intensity invariant. For this data set there is little difference between the distance test and the sequence test.

4.1.2 HMM. For this data set, the HMM algorithm does not achieve the high accuracy rates of the FST NN. Of the features tested, the eight wedge Fourier features worked

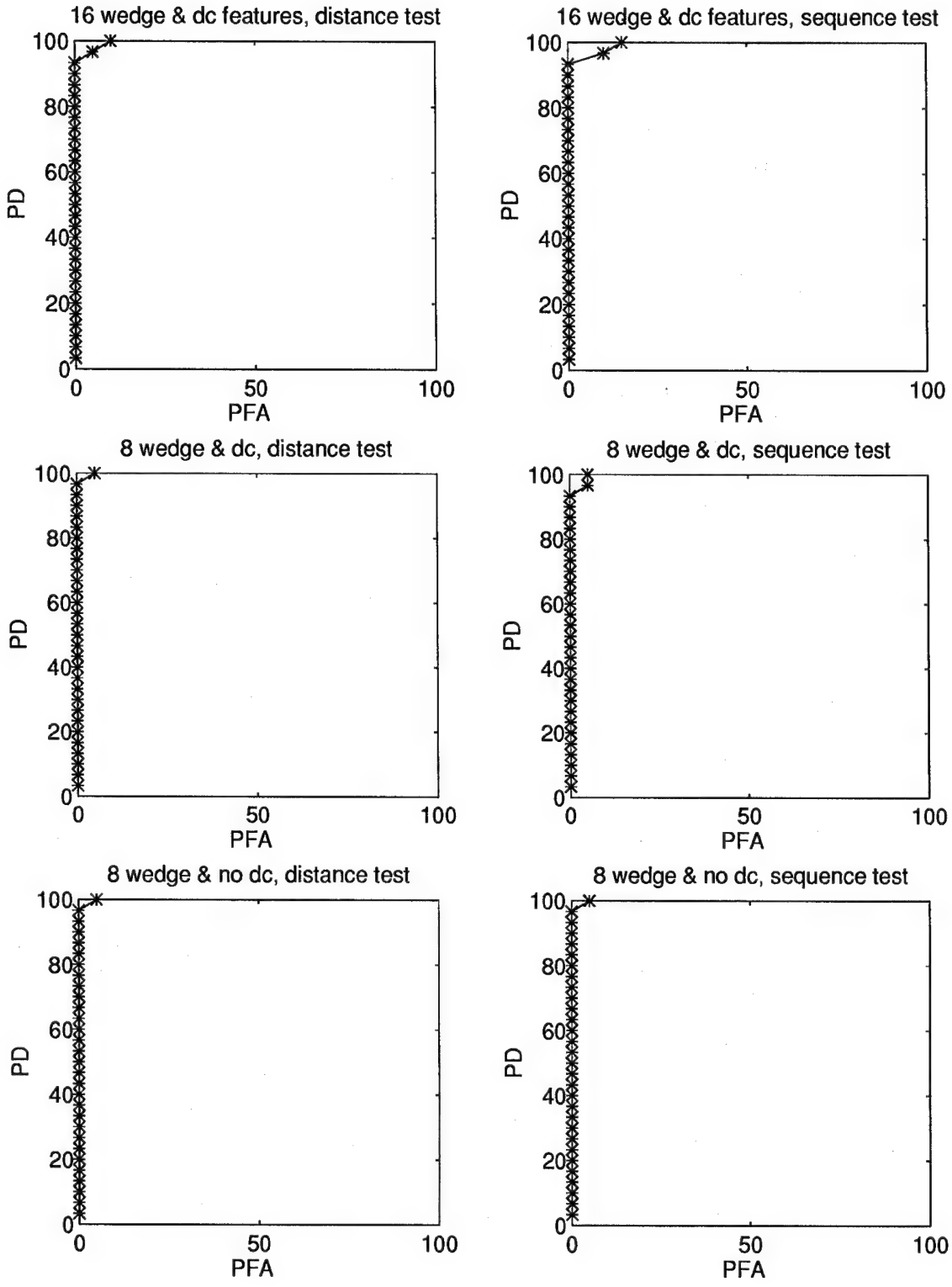

Figure 19. FST NN results on first data set using energy normalized wedge Fourier features. (note: PD = probability of detection, PFA = probability of false alarm)

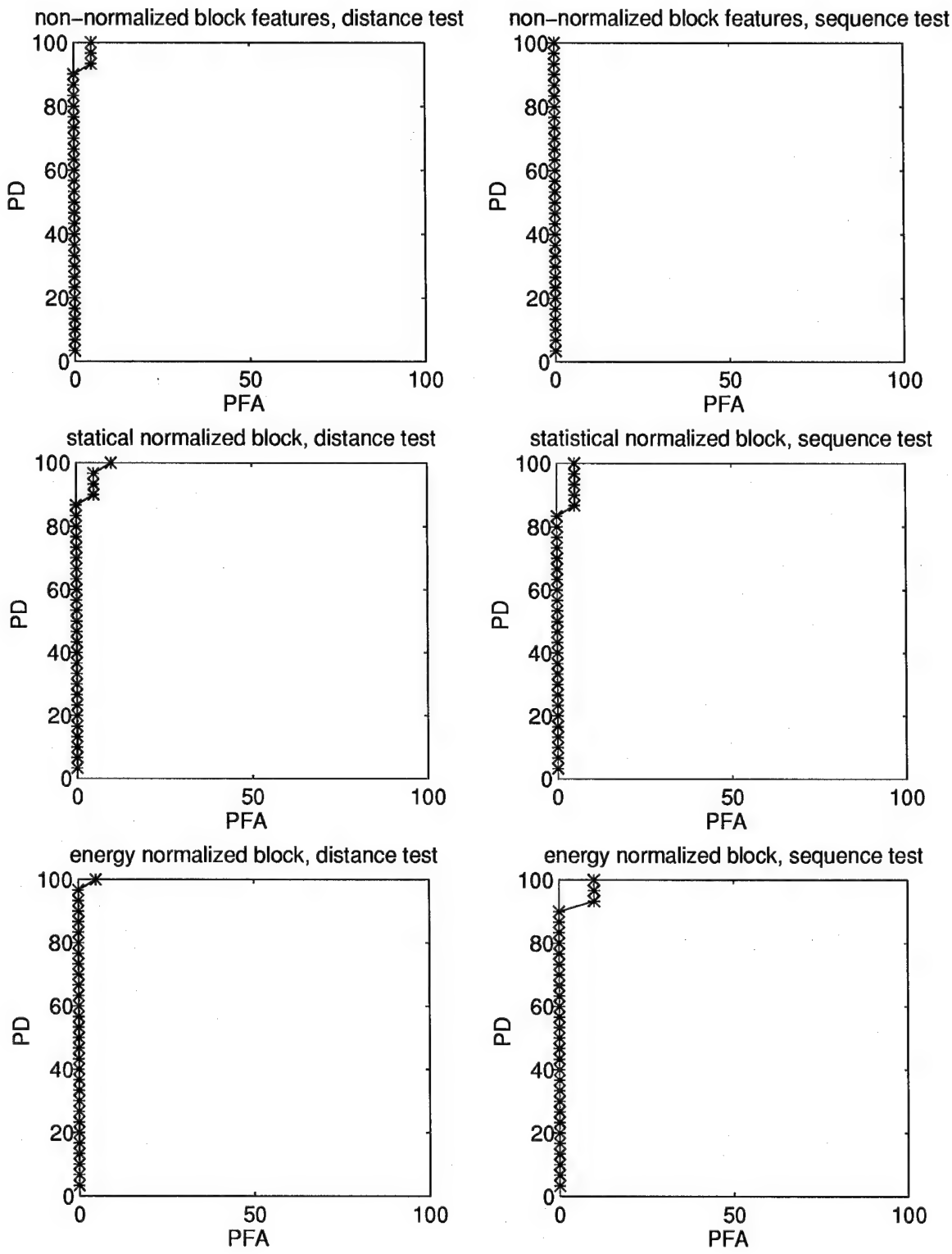

Figure 20. FST NN results on first data set using block Fourier features.

best. Of the types of models tested, the one or two state models performed poorly, the five state model was a little better and the ten state model performed best. A fifteen state left-right model was found to perform slightly below the 10 state model, and more than one Gaussian mixture always improved performance. A four Gaussian mixture was attempted for each case but when the feature vector was large, or many states were used, the algorithm would not work; so a two mixture model was used in those cases. For the 10 state models, a general trend is observed that ergodic models perform better than left-right models (Figures 21 and 22).

Comparing the various features, the energy normalized features seem to work better. The energy normalized block feature results are shown compared to other block features (Figures 23, 24, and 25). The size of the feature vector has a significant impact on HMMs. The Viterbi algorithm does not work when there are too many free parameters. A shorter feature vector will work with less training data, but a longer feature vector requires more training data. An example of this is that four mixture models could not be used with the 18 length block features except for the two state left-right model and one state model; but with the 8 wedge feature vector, four mixtures worked on all left-right models tested.

4.1.3 Observations. For the first data set, the sequence test with non-normalized block features performed best. This was the only test capable of 100% correct classification. However, the energy normalized 8 wedge Fourier features with the FST NN distance test performed well with a much quicker calculation time. The 8 point wedge feature vectors also take less storage than the 18 point block feature vectors.

4.2 Anomaly Detection Tests on the Second Data Set

The second data set was designed to increase classification accuracy. Using only descending passes helps the distance test because there is not the possibility of a test sequence matching up with a training sequence that moves in the opposite direction. Also the image

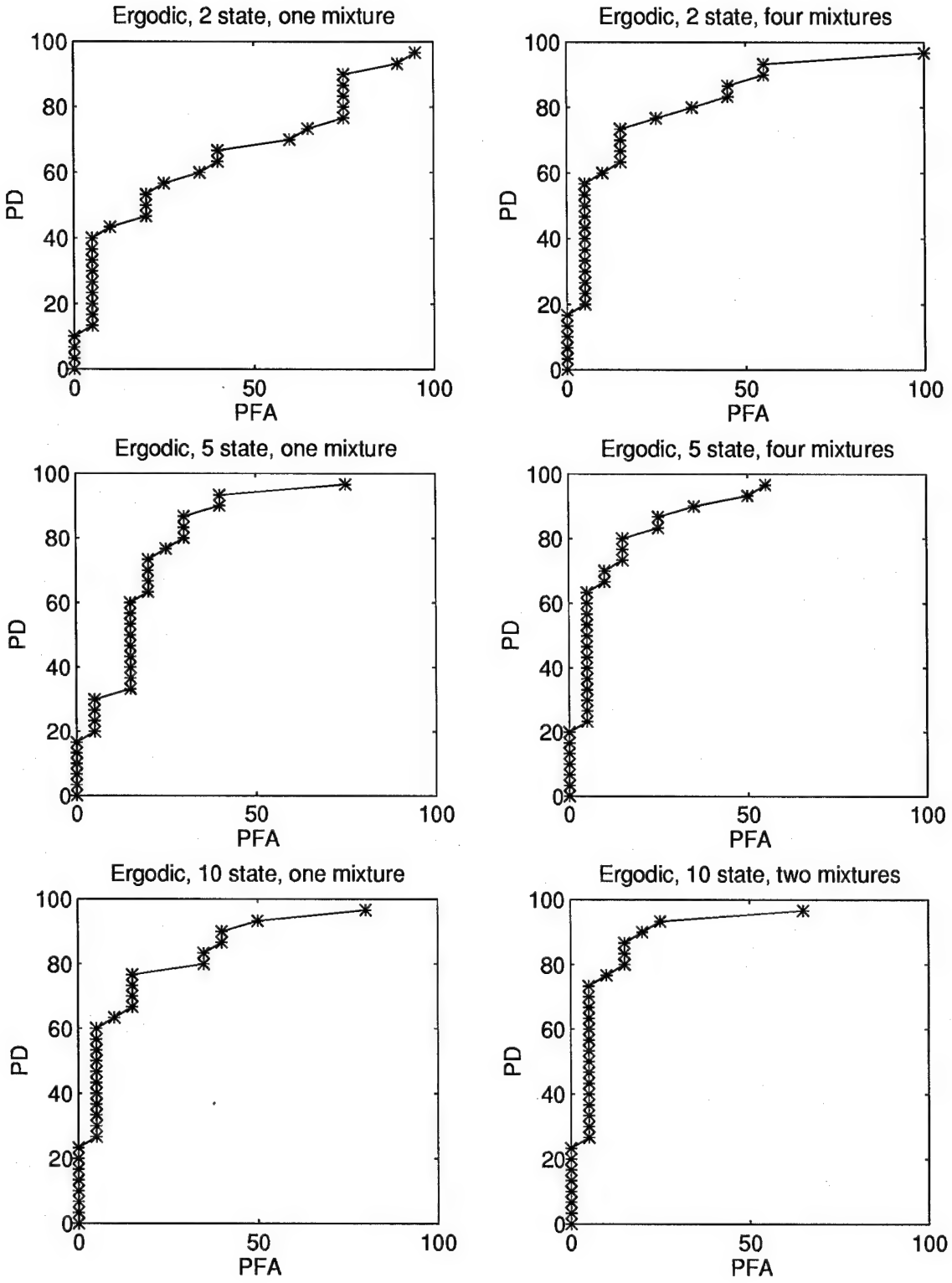

Figure 21. HMM results on first data set using energy normalized 8 wedge Fourier features.

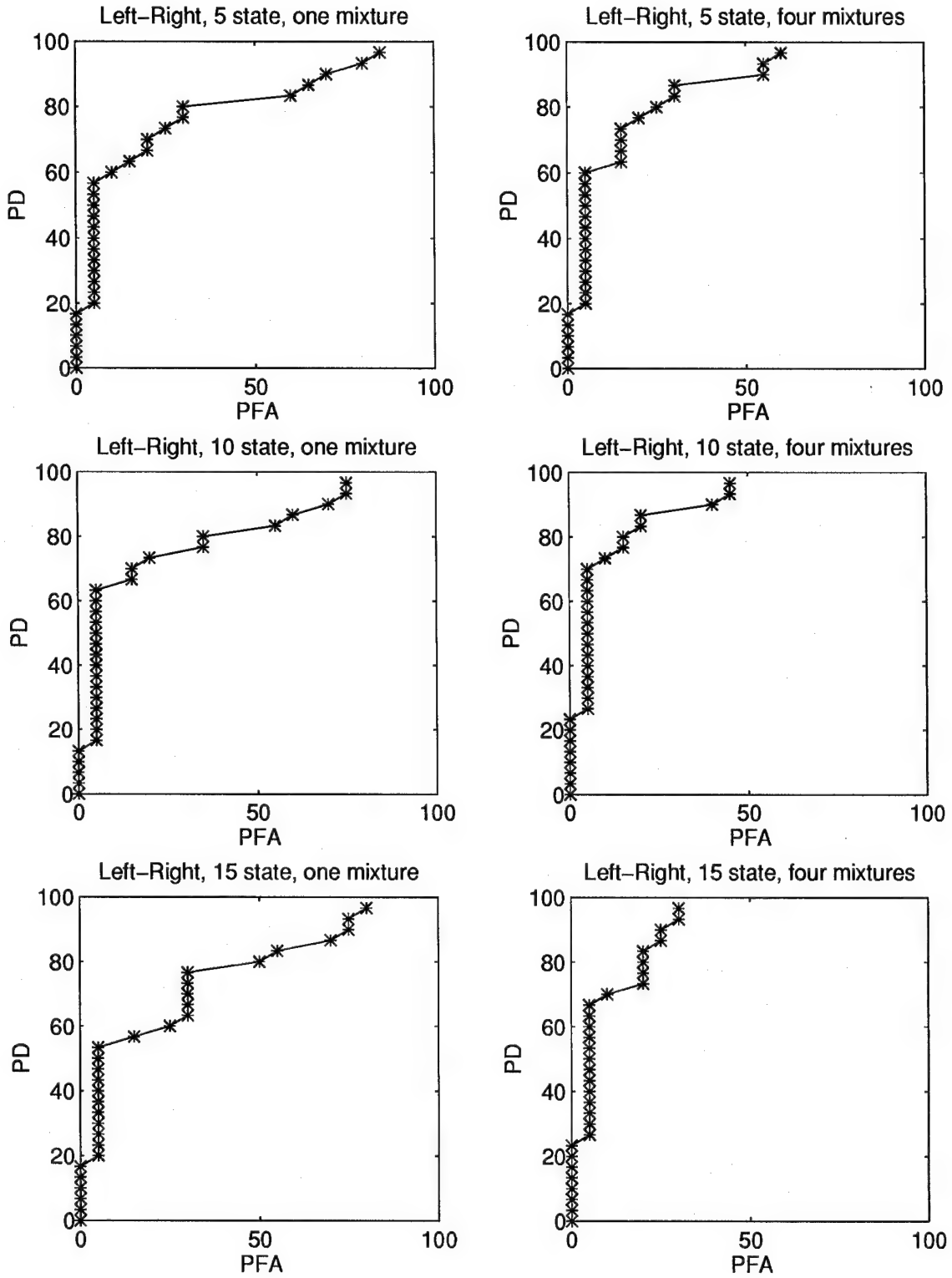

Figure 22. More HMM results on first data set using energy normalized 8 wedge Fourier features.

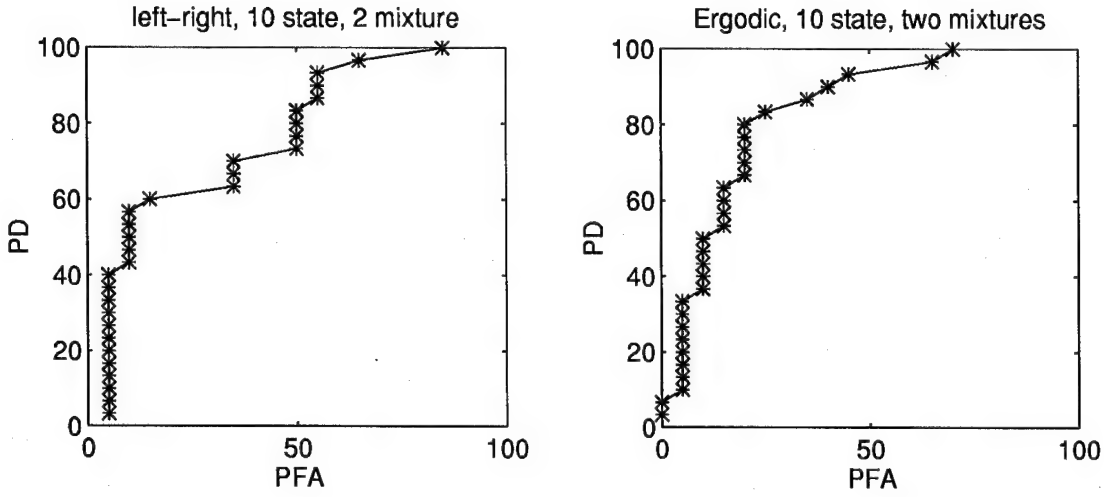

Figure 23. HMM results for non-normalized block features.

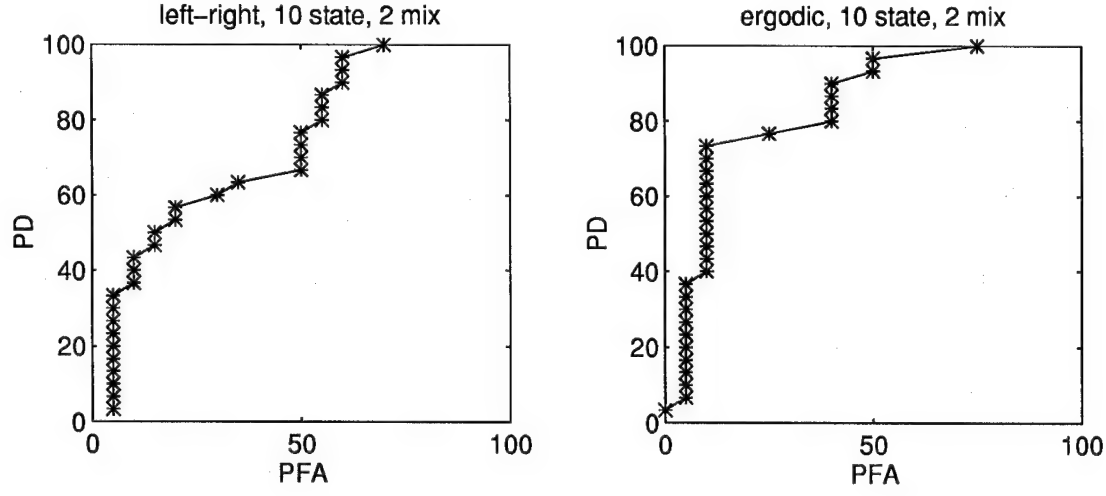

Figure 24. HMM results for statistically-normalized block features.

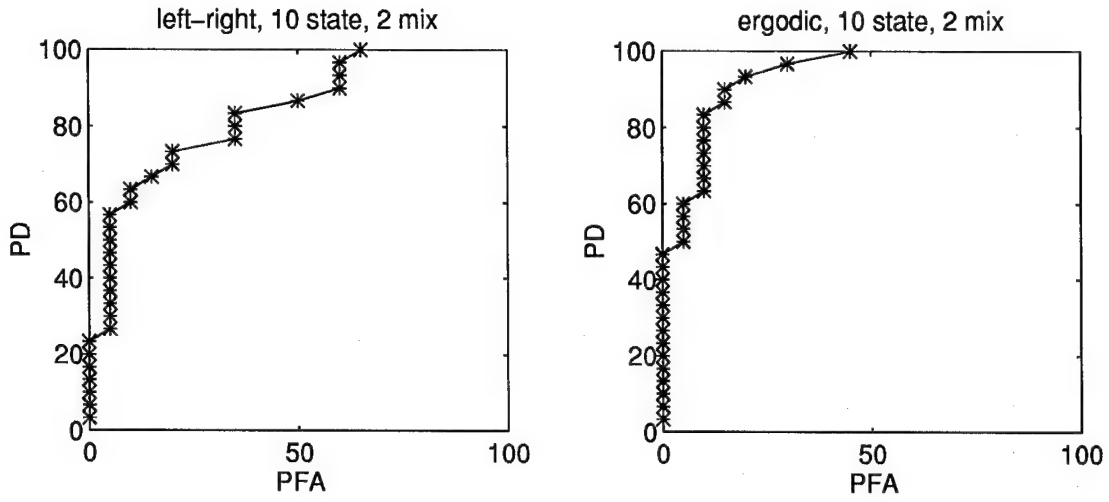

Figure 25. HMM results for energy-normalized block features.

sample spacing was reduced so there is less 'jump' between frames and the space between adjacent sequences is reduced to decrease the difference in aspect between adjacent sequences.

4.2.1 FST NN. The FST NN anomaly detection tests were successful. Of the combinations tested, the best results were achieved by the distance test. The receiver operating characteristic graphs show that 100% detection is possible in every feature space tested. (Figures 26 and 27).

Comparing block Fourier features to wedge Fourier features, there does not appear to be an advantage to either feature set. Eight wedge features outperformed 16 wedge features in the sequence test and using a dc feature did not improve performance.

4.2.2 HMM. For the second data set, the HMM algorithm performs much better than with the first data set. Of the features tested, the block Fourier features worked best (Figures 28 and 29). Of the types of models tested, the one or two state models performed poorly, but the five state model in some cases performs better than the ten state model. The ten and fifteen state left-right models did not perform as well as the ten state ergodic model, and multiple Gaussian mixtures always improved performance compared to single Gaussian

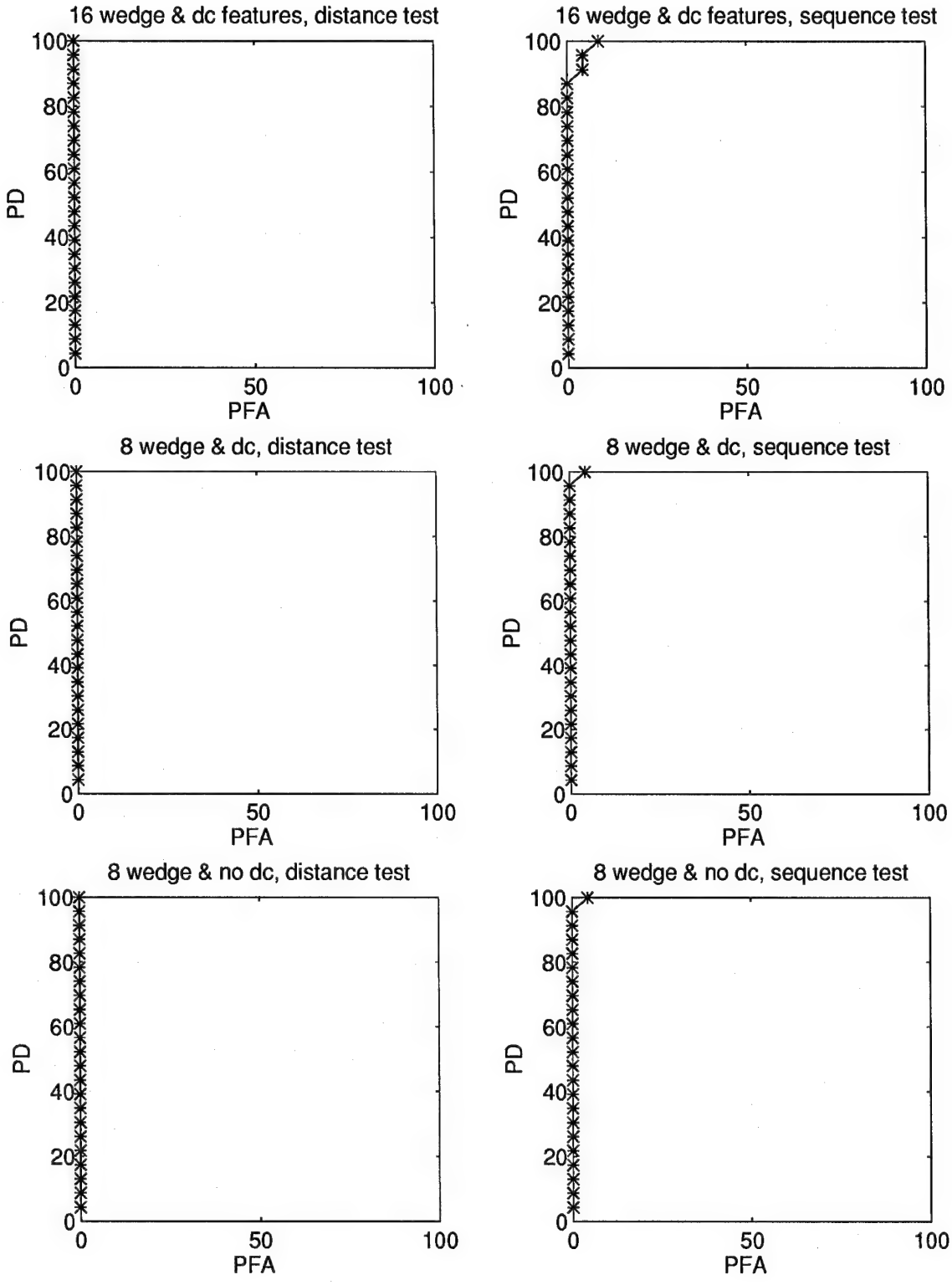

Figure 26. FST NN results on second data set using energy normalized wedge Fourier features.

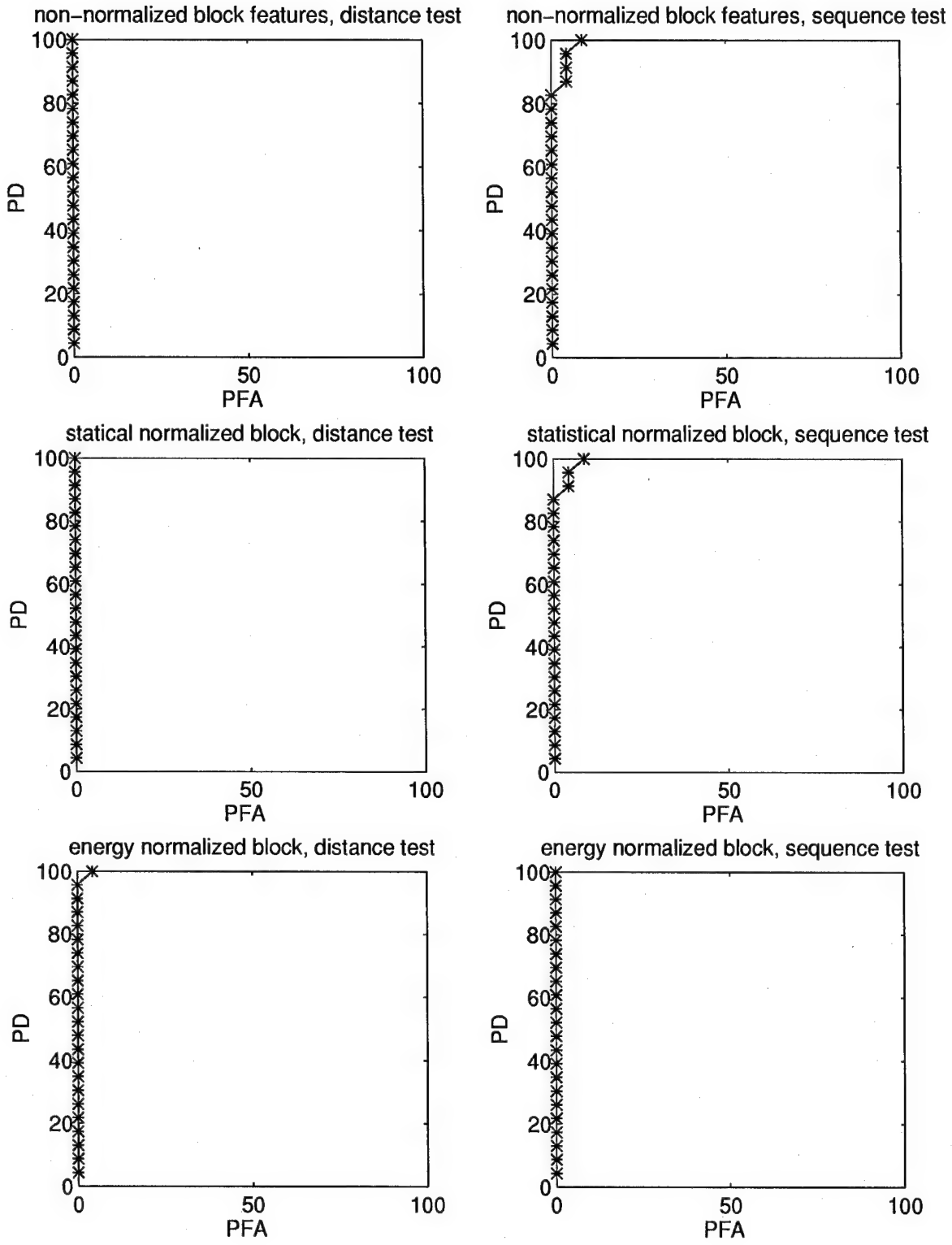

Figure 27. FST NN results on second data set using block Fourier features.

states. A four Gaussian mixture was attempted for each case but in some cases the algorithm would not work due to insufficient data. In these cases two or three Gaussian mixtures were tried. Ergodic models did not work with the energy normalized block features.

4.2.3 Observations. For the second data set the choice of the best algorithm would be a toss up between FST NN and HMM because both are capable of perfect classification. In this experiment, the HMM computes faster, but it is not a fair comparison because the FST NN is run in Matlab.

4.3 Anomaly Detection Tests with Noisy Test Set ($\bar{K}$=100,000)

Shot noise is added to the second data set. The experiments test performance when noise is added to the test data. Adding shot noise had the side-effect of lowering the overall intensity of the images. This is good as it will show how well these algorithms handle the realistic possibility of different intensity levels in the training and test data. The real data provided by AMOS show a definite difference in intensity between various image sequences of the same object.

4.3.1 FST NN. The FST NN is successful testing noisy data with clean training data. For this problem, feature normalization is found to be important. Energy normalized features are most successful. The non-normalized and statistically normalized block features which performed well on the original data, perform poorly on this data set (probably a result of the lower intensity level of the noisy data). Of the combinations tested, wedge features performed best on the distance test but the energy normalized block features performed best with the sequence test. (Figures 30 and 31).

Comparing block Fourier features to wedge Fourier features, there does not appear to be an advantage to either feature set. For the sequence test, eight wedge features outperformed sixteen wedge features and adding a dc feature did not improve performance. The eight wedge features probably performed better because the larger wedge averages out the variations caused by the noise, and the higher resolution of 16 wedges is not required. Although the receiver

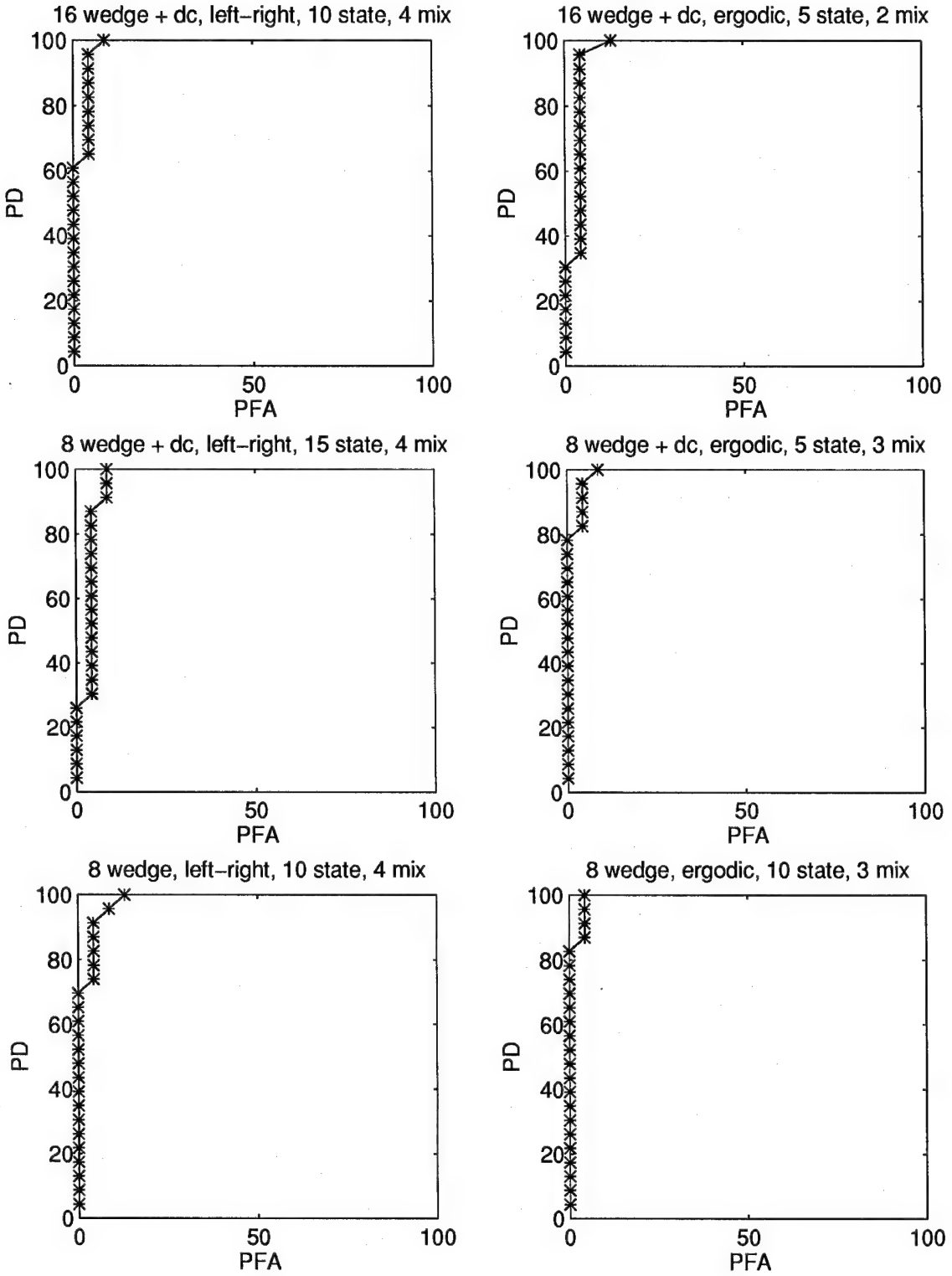

Figure 28. HMM results on second data set using energy normalized wedge Fourier features.

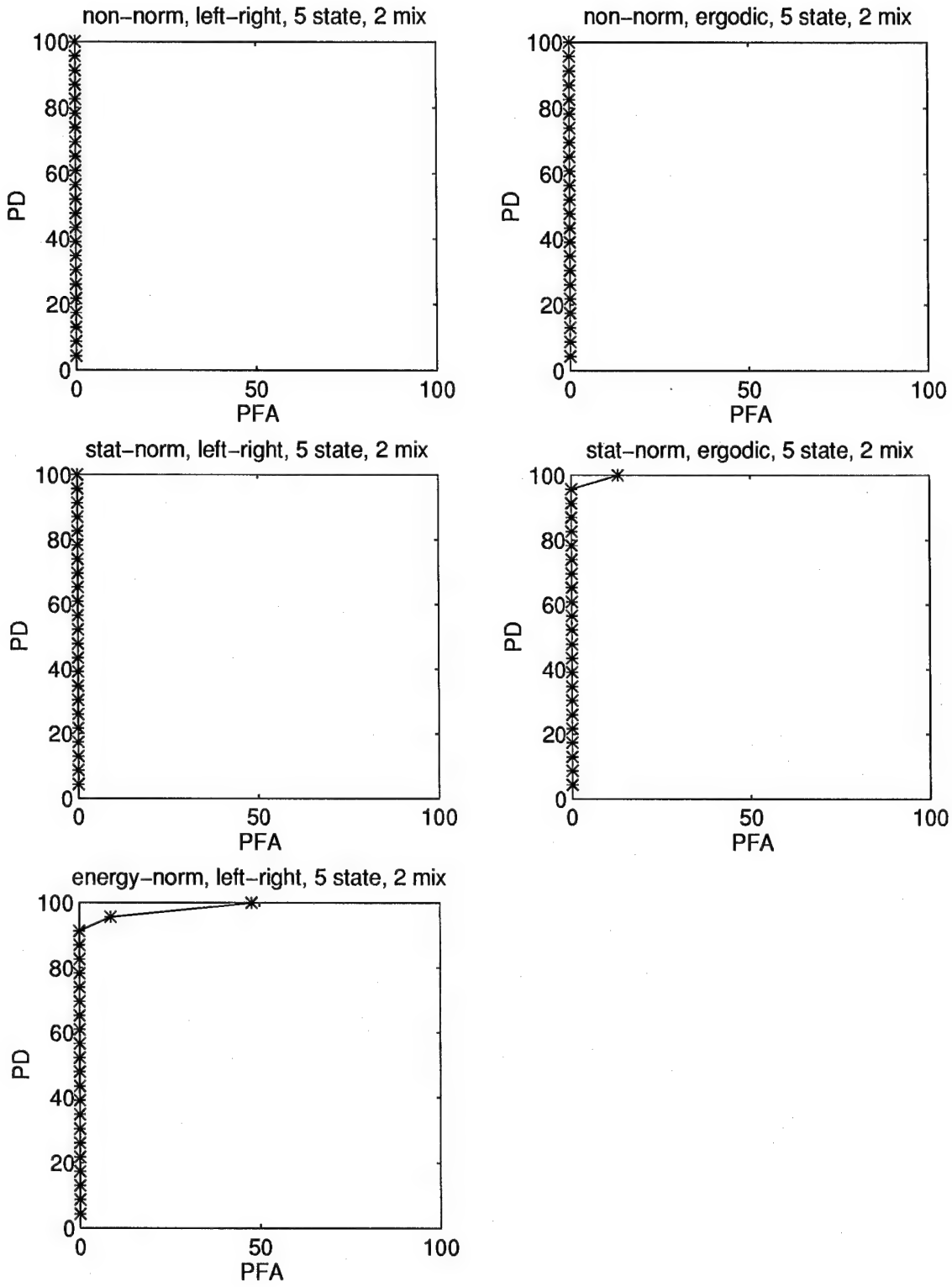

Figure 29. HMM results on second data set using block Fourier features.

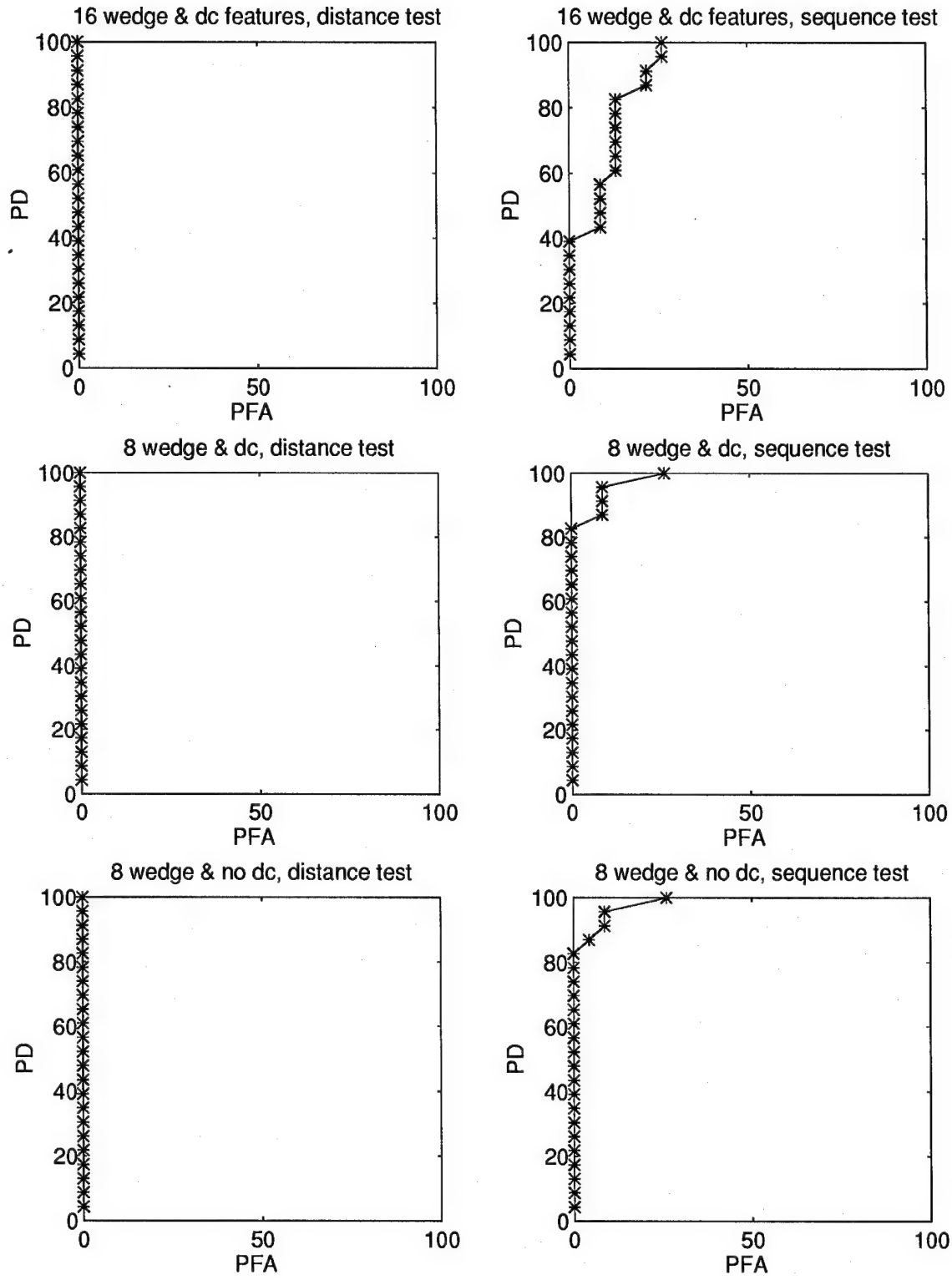

Figure 30. FST NN results on noisy test data ($\bar{K}$=100,000) using clean training data and energy normalized wedge Fourier features.

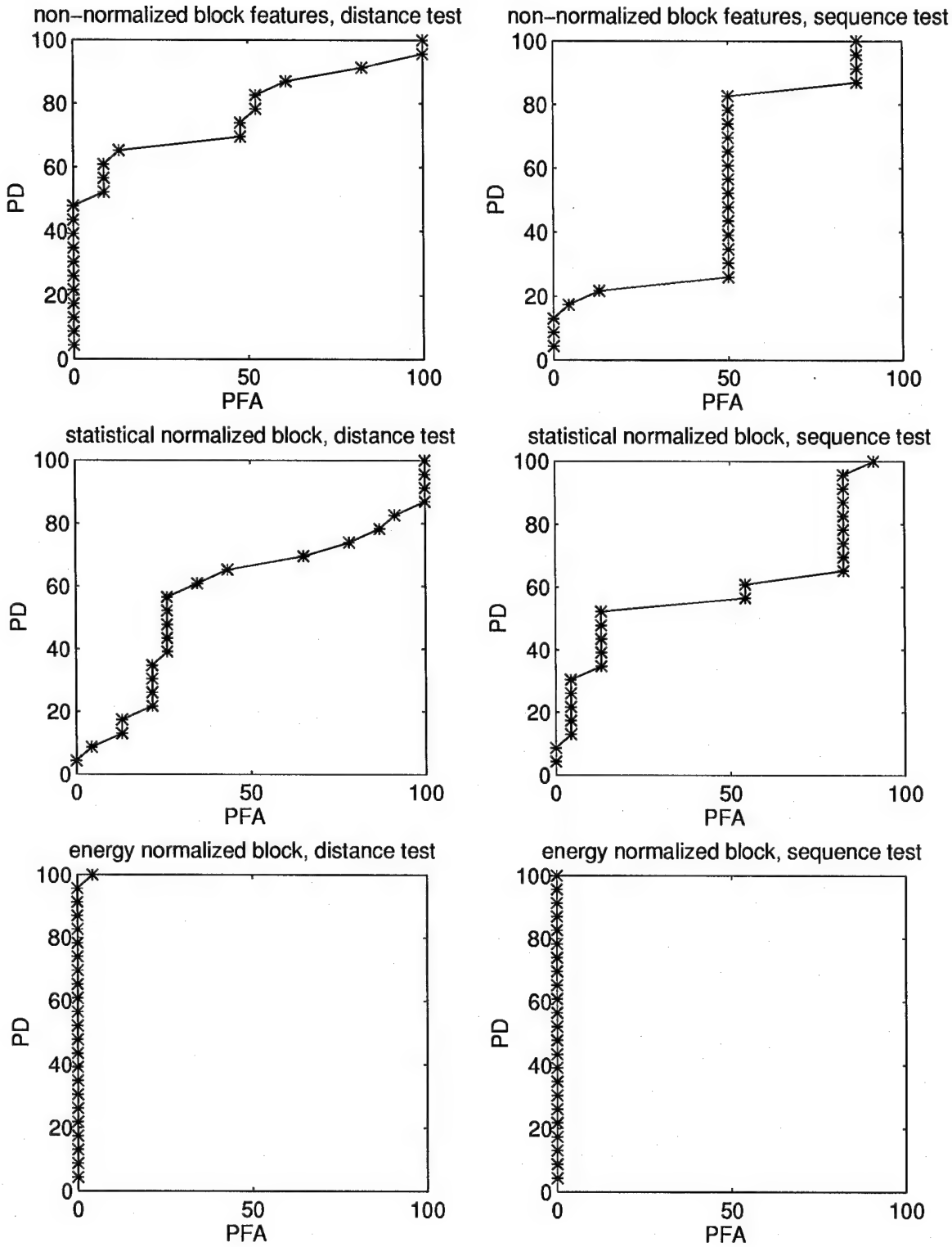

Figure 31. FST NN results on noisy test data ($\bar{K}$=100,000) using clean training data and block Fourier features.

operating characteristic graphs look the same, the results using wedge features with noisy data were different from the results using original data, but the energy normalized block features produced the same results in both noisy and original data sets (Figure 32). Tests with increased shot noise ($\bar{K}$=15,000) showed similar results.

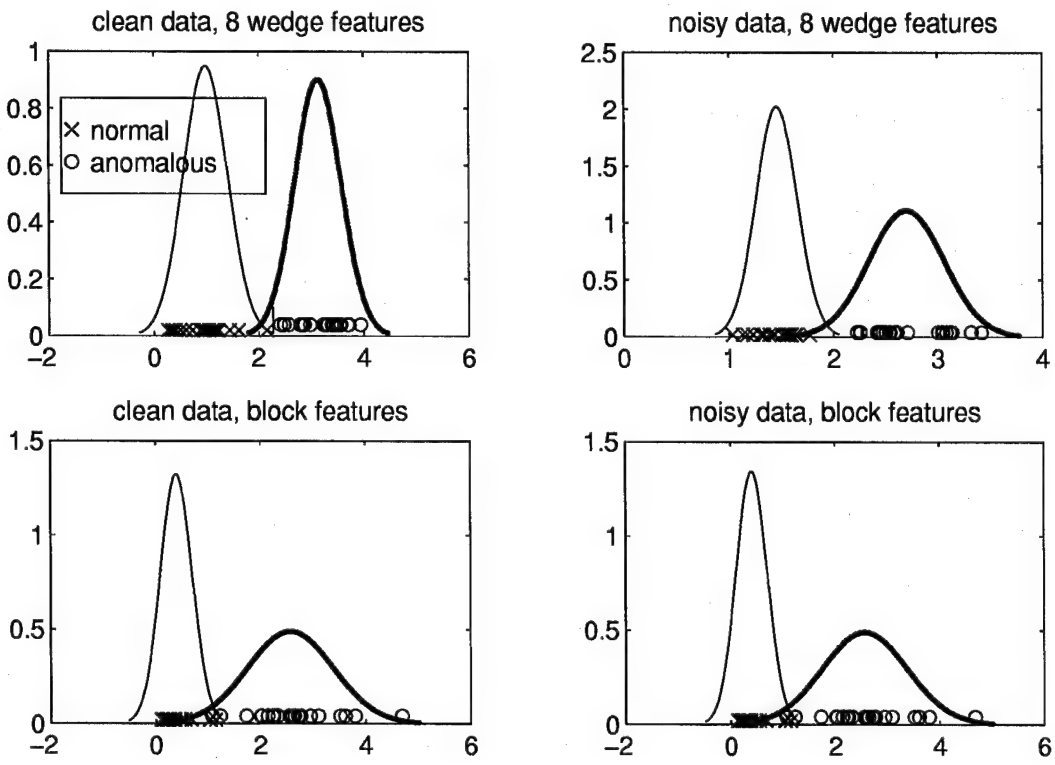

Figure 32. FST NN results on clean versus noisy ($\bar{K}$=100,000) test data set using energy normalized features and distance test. The block feature results are practically the same whether clean or noisy test data is used.

4.3.2 HMM. Some unexpected results were discovered when testing noisy data with the HMM. For the tests using clean data, the ergodic models usually performed equal or better than left-right models, but for the noisy data the left-right models worked much better (Figures 33 and 34). All features tested had similar performance.

The distribution of the results of this test were interesting. The 8 wedge feature space had one anomalous class outlier, but the 8 wedge with dc feature space had one normal class

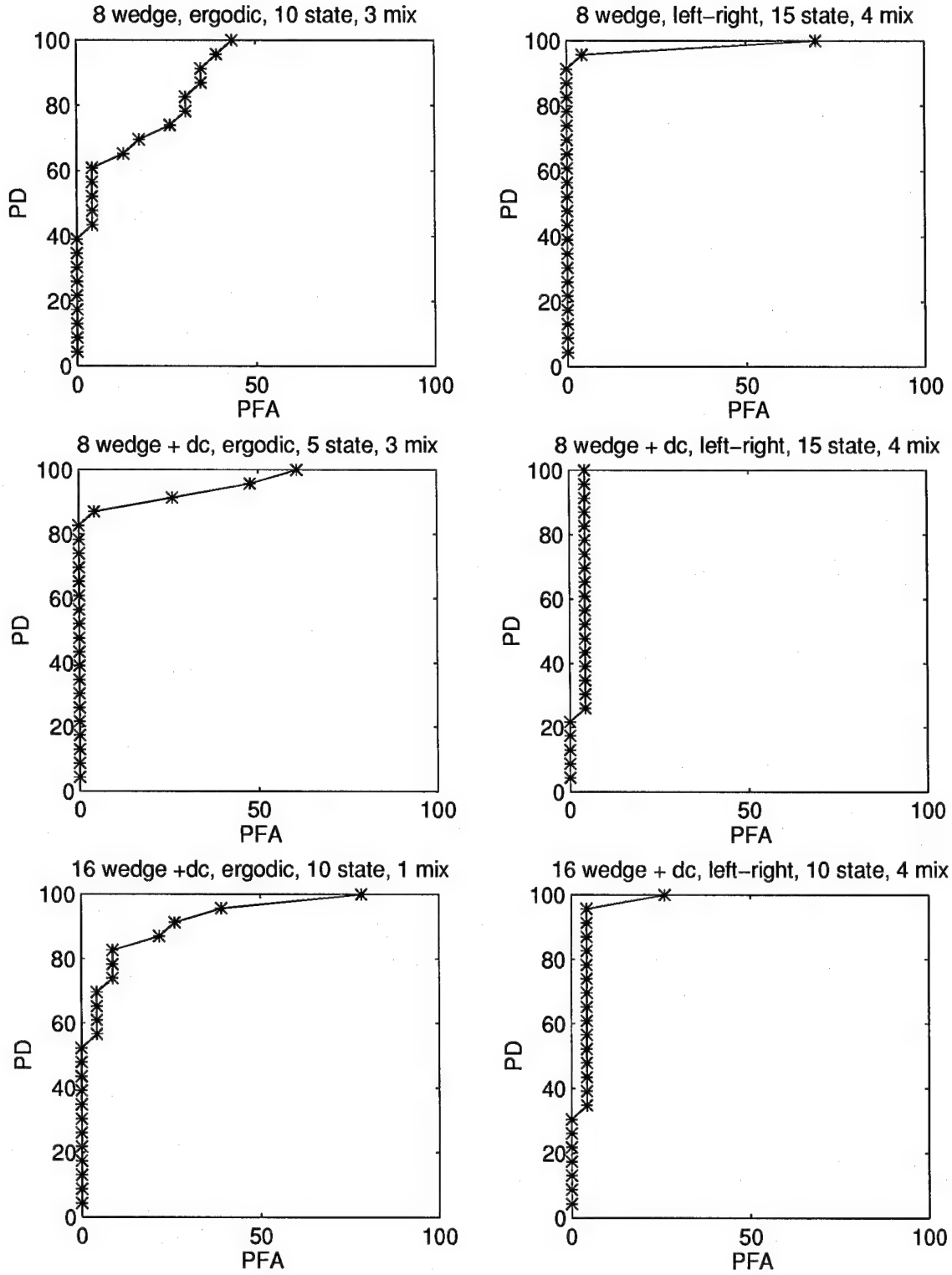

Figure 33. HMM results on noisy ($\bar{K}$=100,000) data set.

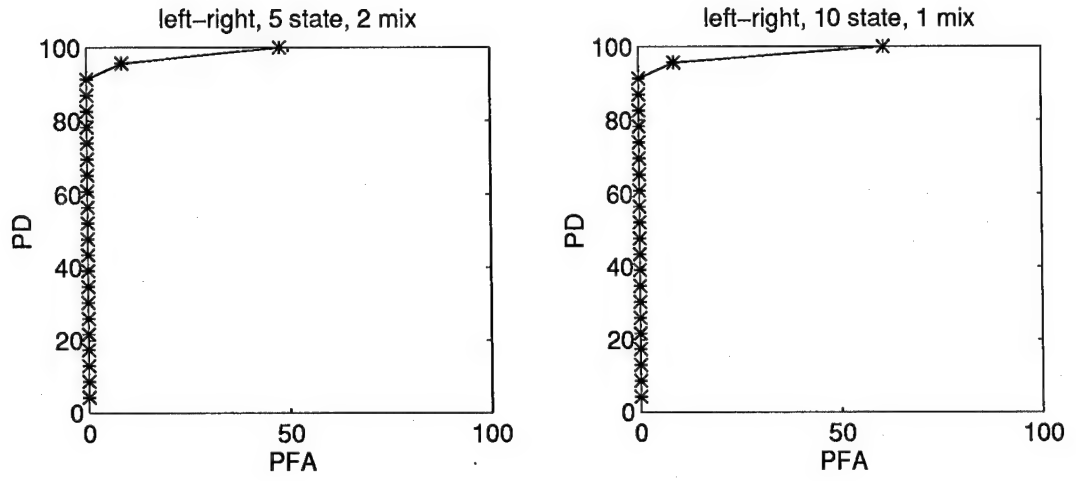

Figure 34. HMM results on noisy ($\bar{K}$=100,000) data set using energy normalized block features.

outlier (Figure 35). For the non-normalized and statistically normalized block features, the distributions show there is a large difference between the training and test sets (Figure 36).

4.3.3 Using Noisy data in Training Set. One would expect that the results on noisy test data would improve if noisy training data are used ($\bar{K}$=100,000 for all results in this section). Results show there is not much difference when using energy normalized features (Figures 37, 38, and 39). The features that are not energy normalized perform much worse with clean training data because of the difference in intensity levels between clean data and noisy data (Figures 40).

4.3.4 Observations. Overall, the FST NN distance test had the least degradation when noisy data are introduced. The energy normalized features perform best because they are intensity invariant. The 8 wedge Fourier features are the best choice for computational efficiency.

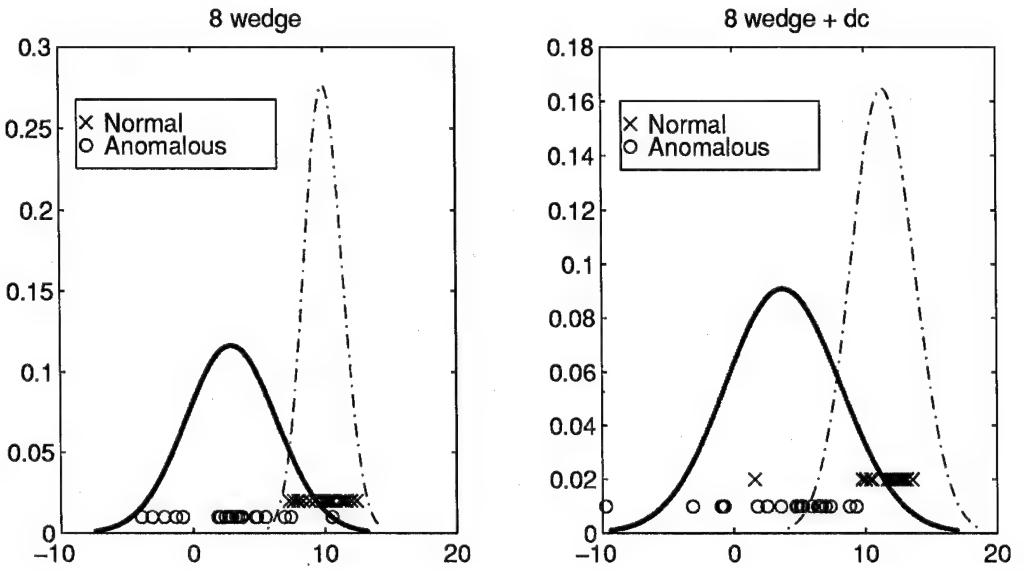

Figure 35. Distribution of values for left-right, 15 state, 4 mix model, noisy ($\bar{K}$=100,000) test data.

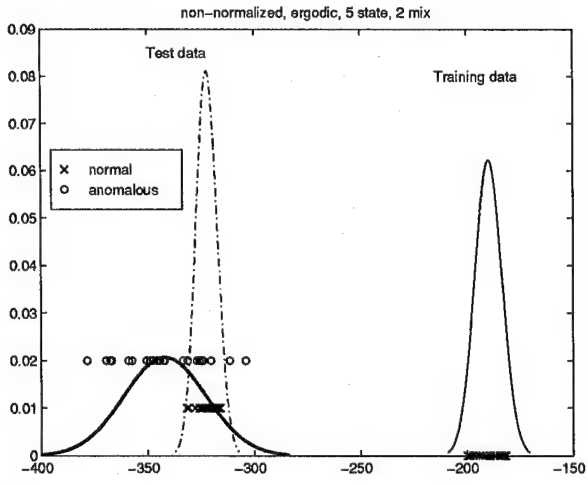

Figure 36. Distribution of values from HMM on non-normalized block features using noisy ($\bar{K}$=100,000) test data and clean training data.

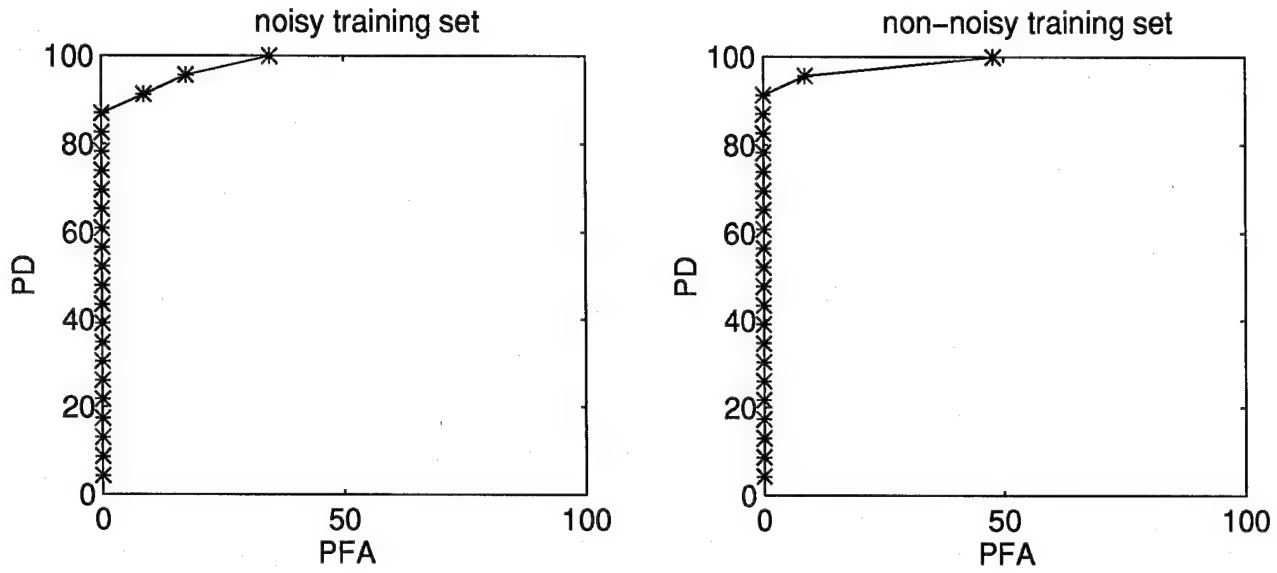

Figure 37. Comparison of HMM results on noisy ($\bar{K}$=100,000) vs clean training data with energy normalized block features and left-right, 5 state, 2 mix model.

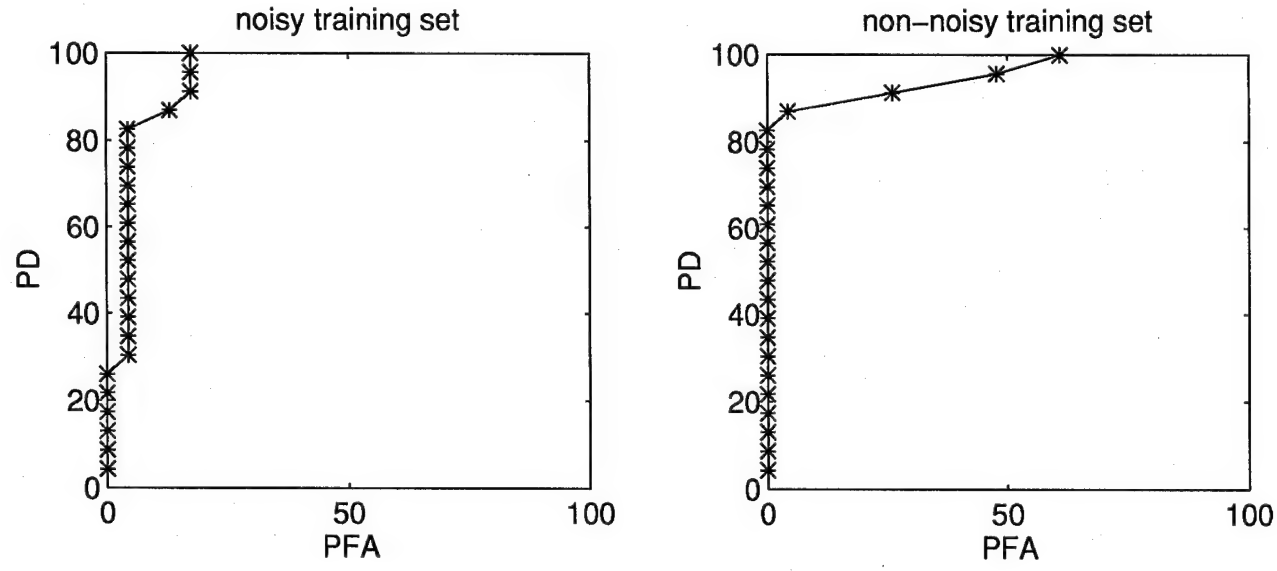

Figure 38. Comparison of HMM results on noisy ($\bar{K}$=100,000) vs clean training data with energy normalized 8 wedge + dc features and ergodic, 5 state, 3 mix model.

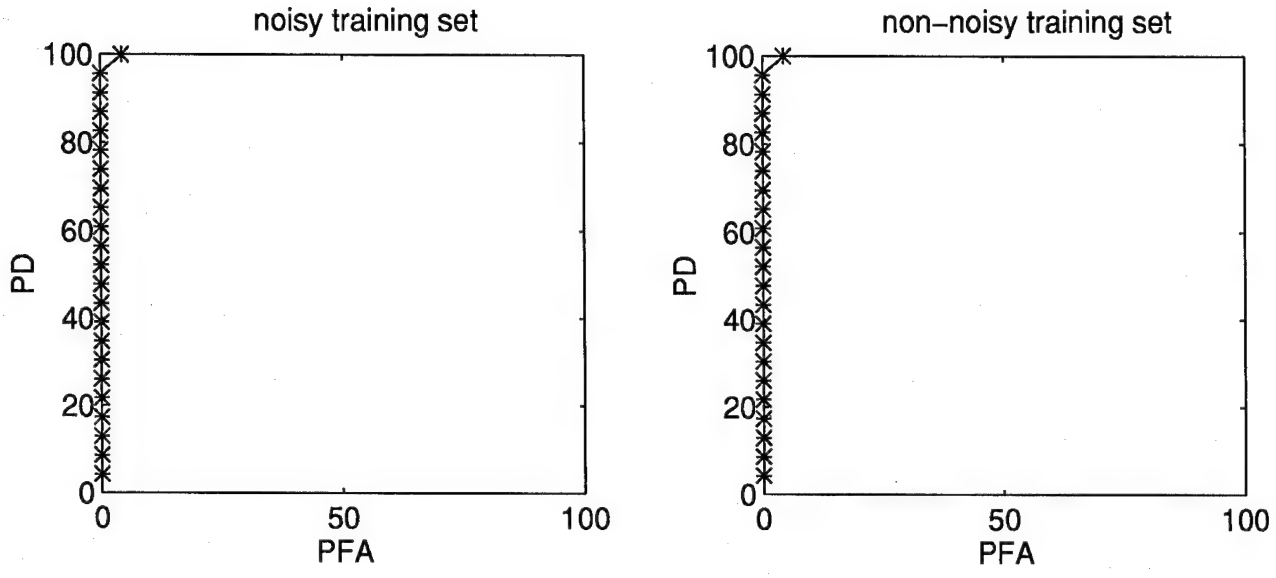

Figure 39. Comparison of FST NN results on noisy ($\bar{K}$=100,000) vs clean training data with energy normalized block features.

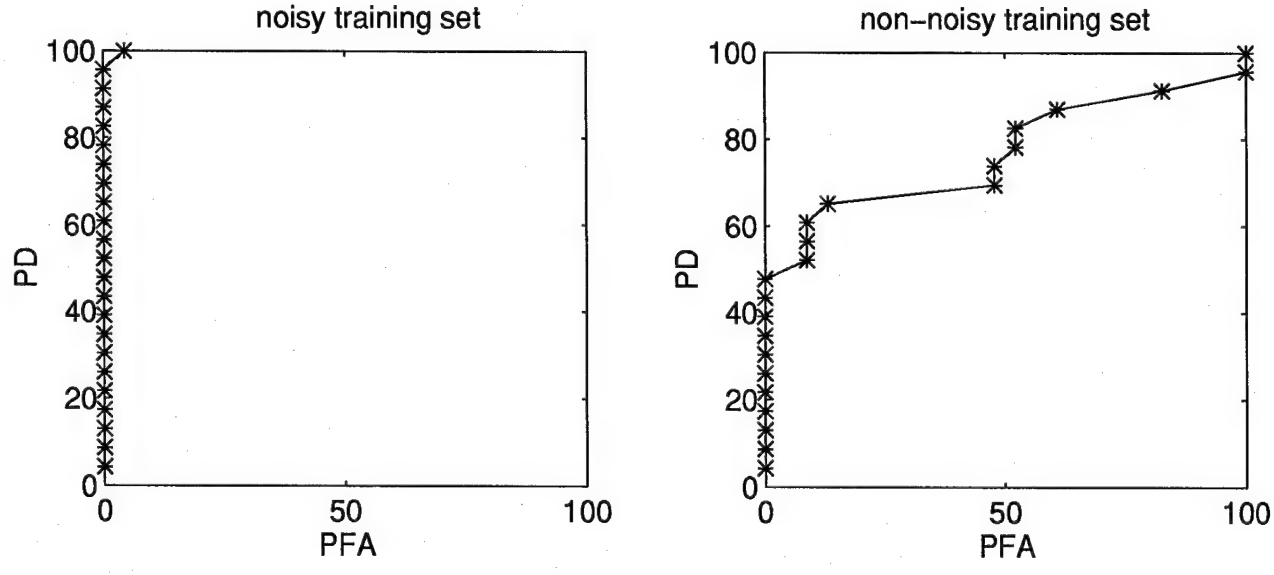

Figure 40. Comparison of FST NN results on noisy ($\bar{K}$=100,000) vs clean training data with non-normalized block features.

4.4 *Anomaly Detection Tests on Noisy Test Set ($\bar{K}$=15,000)*

The previous sections show that some options are not worth pursuing. Table 1 summarizes the options that are and are not evaluated in this section. Non-energy normalized features are not considered because of their poor performance on tests where the noise levels differ between training and test sets. Histogram equalization would compensate for this problem but is not used because it is much more efficient to energy normalize a feature vector than to process an entire image.

Table 1. Summary of results from previous sections

Combinations eliminated from consideration		
Classifier	Feature	Why
HMM	all	For each feature tested so far, the FST NN performed equal or better
FST NN both	stat- and non-normalized block	When noise level differs between test and train, performance is terrible
FST NN sequence	wedge	P_{FA} exceeds 25% when P_D is 100%
Remaining combinations		
FST NN distance	Energy normalized features (wedge and block)	
FST NN sequence	Energy normalized block features	

4.4.1 Tests With Training Set Noise of $\bar{K}$=100,000. All remaining combinations achieve the same ROC graph results as they do at lower noise levels. The distance test with wedge features, and sequence test with energy normalized block features achieve perfect separation and the distance test with energy normalized block features gets one false alarm (4% P_{FA}) with 100% anomaly detection (P_D) — the same ROC as shown in figure 39.

4.4.2 Tests With Training Set Noise of $\bar{K}$=15,000. With a noise level of $\bar{K}$=15,000 in the test and training sets, the class separation performance decreases slightly for all combinations except the energy normalized block features which remains at 100% P_D and 4% P_{FA} as in figure 39.

4.5 Classification Thresholds

The ROC graphs show that the FST NN with energy normalized features can separate the classes. The difference in capability between wedge and block features is found when applying the classification threshold or decision boundary. Four methods are tested and results are summarized in tables 2, 3, and 4. The wedge features are all energy normalized and produce similar results, so the comparisons in this chapter use 16 wedge (with dc) to represent the wedge features.

Table 2. Threshold results for FST NN distance test with 16 wedge + dc Fourier features (P_D/P_{FA} in percent))

Threshold type	Data Set 1 (DS1)	Data Set 2 (DS2)	DS2, clean trn, $\bar{K}$=100,000 test	DS2, $\bar{K}$=100,000 trn, $\bar{K}$=100,000 test	DS2, $\bar{K}$=100,000 trn, $\bar{K}$=15,000 test	DS2, $\bar{K}$=15,000 trn $\bar{K}$=15,000 test
borrowed	97 / 10	100 / 4	100 / 0	91 / 0	39 / 0	0 / 0
mix-up	76 / 0	100 / 0	100 / 0	100 / 9	100 / 30	100 / 9
mismatch	N/A	N/A	N/A	N/A	N/A	N/A
leave-one	90 / 0	100 / 0	100 / 0	97 / 0	97 / 0	93 / 0

Table 3. Threshold results for FST NN distance test with energy normalized block Fourier features (P_D/P_{FA} in percent)

Threshold type	Data Set 1 (DS1)	Data Set 2 (DS2)	DS2, clean trn, $\bar{K}$=100,000 test	DS2, $\bar{K}$=100,000 trn, $\bar{K}$=100,000 test	DS2, $\bar{K}$=100,000 trn, $\bar{K}$=15,000 test	DS2, $\bar{K}$=15,000 trn $\bar{K}$=15,000 test
borrowed	100 / 5	100 / 13	100 / 13	100 / 13	100 / 13	100 / 13
mix-up	70 / 0	96 / 0	96 / 0	96 / 0	96 / 4	96 / 4
mismatch	N/A	N/A	N/A	N/A	N/A	N/A
leave-one	100 / 5	100 / 13	100 / 13	100 / 13	100 / 13	100 / 13

Table 4. Threshold results for FST NN sequence test with energy normalized block Fourier features (P_D / P_{FA} in percent))

Threshold type	Data Set 1 (DS1)	Data Set 2 (DS2)	DS2, clean trn, $\bar{K}$=100,000 test	DS2, $\bar{K}$=100,000 trn, $\bar{K}$=100,000 test	DS2, $\bar{K}$=100,000 trn, $\bar{K}$=15,000 test	DS2, $\bar{K}$=15,000 trn $\bar{K}$=15,000 test
borrowed	90 / 0	96 / 0	96 / 0	96 / 0	100 / 4	96 / 0
mix-up	90 / 0	65 / 0	65 / 0	65 / 0	65 / 0	78 / 0
mismatch	97 / 10	91 / 0	91 / 0	91 / 0	91 / 0	91 / 0
leave-one	97 / 10	100 / 4	100 / 4	100 / 4	100 / 4	100 / 4

4.5.1 Borrowed Anomalous Method. This approach assumes that anomalous data can be acquired from another data set. Anomalous data from the first data set is used to find a threshold for the second data set, and vice versa. This works well on clean data but does not work well on the wedge features with noisy data. The results on energy normalized block features, however, are not degraded by shot noise. The results are shown with a threshold at 90% of the minimum value of the borrowed anomalous results. This method produces a fairly consistent threshold for the block features but it degrades with noise when using wedge features.

4.5.2 Mix-Up Method. Another approach is to make anomalous sequences by mixing feature vectors from various training sequences. This method worked pretty well and it also has the advantage of not requiring any data outside the training set. Once again the energy normalized block features are not degraded as much as the wedge features. This method is consistent across noise levels, but inconsistent between data sets one and two.

4.5.3 Forced Mismatch Method. A third approach, that only applies to the sequence test (not possible with the distance test), is to force a mismatch within the training set then use the resulting sequence statistics as anomalous data. This method worked pretty well on energy normalized block features and it also has the advantage of not requiring any data outside the training set. This method is consistent across data sets and noise levels, but only works with the sequence test.

4.5.4 Leave-One-Out Method. A fourth approach is based strictly on normal training data and uses the mean times 1.3 of leave-one-out results for the classification threshold. This method worked pretty well and it also has the advantage of not requiring any data outside the training set. With this threshold, the wedge features and energy normalized block features performed well. This method produces consistent results across both data sets and the various noise levels. It works well on the distance test as well as the sequence test.

4.5.5 Summary of Threshold Tests. The leave-one-out method produces the best results across both data sets and the various noise levels. It works on wedge features as well as block features and on the distance test as well as the sequence test. The block features do not degrade as noise levels are increased. However, it is not perfect. Data set one has a higher level of missed anomalies and data set two has a higher level of false alarms. This indicates that the leave-one-out threshold is sensitive to the spacing of the data in the training set.

4.6 Summary of Results

The winner is — FST NN sequence test with energy normalized block Fourier features using leave-one-out threshold method.

There are trade-offs with this decision, however. The sequence test requires each sequence of images to be the same length with the same time between frames, but the distance test does not have this restriction. The block features are not scale invariant, so the wedge features would be preferable in cases where scale invariance is required. Also, if varying noise levels are not a problem, the 8 wedge features will compute much faster because the feature vector is less than half the length of the block feature vectors.

V. Conclusion and Recommendations

5.1 Conclusions

The basic findings of this research are:

- The FST NN performs as well or better than the HMM in all tests.
- Energy normalization performs much better when noise level is not consistent (varying image intensity levels was probably the reason)
- The leave-one-out derived threshold works well on distance and sequence tests, and across the various data sets and noise levels.
- Both wedge and block Fourier features can separate the data into classes, but results are more consistent with energy normalized block features as the noise level increases.

The FST NN sequence test with energy normalized block Fourier features using leave-one-out threshold method produced the best consistent results across the data sets and noise levels (100% P_D / 4% P_{FA}, data set 2, all noise levels). The most important advantage of the FST NN is performance. This is most evident in the first data set and the noisy data sets. The energy normalized features have a definite advantage when classifying a test set where the noise level is different from the training set. There are some limitations imposed by this decision, however. The sequence test requires each sequence of images to be the same length with the same time between frames and block features are not scale invariant.

In addition to accuracy, another consideration that favors the FST NN over the HMM is simplicity. For the FST NN, the only decision to be made is where to place the classification threshold and this can be determined from the training data using leave-one-out or simulated anomalous data. The operator can adjust this value according to the relative importance of missed detections versus false alarms. The HMM, on the other hand, requires a determination of what type of model would work best. If too many states or mixtures are used, the algorithm will not work, but if too few states or mixtures are used, the classification accuracy will suffer.

The best model depends on the type of data being evaluated, so new applications would require some experimentation to find the best model.

If operational flexibility is important, the FST NN distance test with wedge features is the best choice. The results at lower noise levels are excellent. The energy normalized wedge features perform well on clean data (100% P_D / 0% P_{FA}, data set 2) but degrade slightly as noise is added. The 8 wedge features compute much faster because the feature vector is less than half the length of the block feature vector, and wedge features are scale invariant. Also, the distance test is more flexible because it does not require a fixed number of images per sequence or a fixed interval between images as the sequence requires. However, for a problem where the sequence of images may appear in reverse order, the distance test would not detect this difference, and the sequence test would be more appropriate.

The FST NN is a very effective algorithm for spatio-temporal pattern recognition. On the more difficult first data set, the FST NN achieved 100% anomaly detection with one false alarm (5%) using block features. On the second data set the FST NN achieved a perfect 100% anomaly detection with no false alarms using wedge features. The FST NN is simple to implement and does not require iterative training or hidden layers like other neural nets. The HMM is also an effective algorithm for spatio-temporal pattern recognition, but it is not as accurate as the FST NN for the SOI problem. Also, it is more difficult to implement, requiring more choices to find the optimal model (number of states, ergodic versus left-right, and initial weights must be selected).

5.2 Possible applications of this research

The nadir pointing sun-synchronous satellites, as shown in this thesis, are well suited for pattern recognition because the illumination is consistent for a given aspect view. Geosynchronous satellites are also well suited for pattern recognition since the azimuth and elevation remain basically constant, but the training data would need to cover a full year to cover the full range of illumination conditions. Other SOI problems, where the aspect view and illumination change in a predictable manner, could also be approached with spatio-temporal pattern

recognition. Satellites that are highly maneuverable or change configurations in an irregular manner would not be good candidates for these algorithms.

5.3 Bottom Line

Many pattern recognition gurus are looking for more complex ways to build classifiers. I am glad that Neiberg and Casasent developed the FST NN. This proves that sometimes a new and better method is less complex than the old. The FST NN is an effective algorithm for spatio-temporal pattern recognition. It is simple to implement and does not require iterative training or hidden layers like other neural nets. It does not require selecting the best model or Gaussian mixtures like HMMs. Is it a real neural net? Maybe not, but it works.

Appendix A. FST NN Distance Calculations

Each image sequence is represented as a trajectory in feature space. Each vertex in a trajectory represents an image.

$$\text{each sequence in training set} \Longrightarrow trajectory = X$$

$$\text{each image in training sequence} \Longrightarrow vertex = x_i$$

For each trajectory being tested, the closest training trajectory must be found. The test trajectory distance is the sum of the distance from each vertex of the test trajectory, to the nearest part of the training trajectory (Figure 41). Each trajectory being tested must be

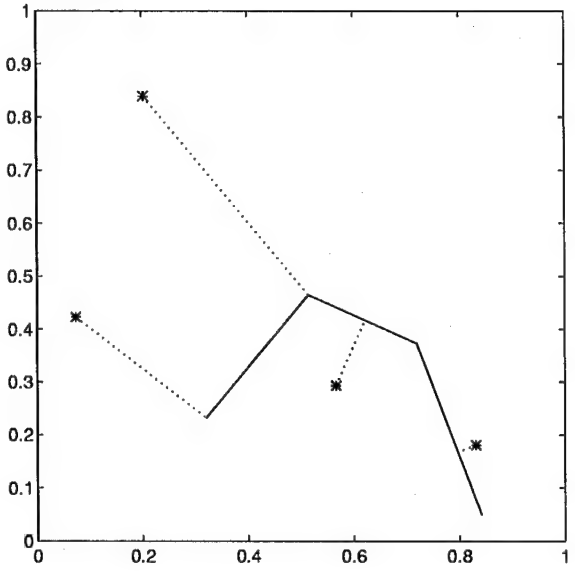

Figure 41. Example of distance from test trajectory to training trajectory (sum of dotted lines).

compared to all trajectories in the database to find the closest one.

Each segment of the trajectory is represented by a length l_i and direction v_i.

$$l_i = \|x_{i+1} - x_i\|$$

$$v_i = (x_{i+1} - x_i)/l_i$$

The inner products are also needed for distance calculations. The inner products are denoted by *c*.

$$c_{i,i+1} = x_i \cdot x_{i+1}$$

All calculations so far are based only on the training set. These are computed in advance.

Now we will consider the sequence being tested. Each test sequence will be represented as a trajectory denoted as P with the vertexes denoted as p.

$$\text{sequence to be tested} \Longrightarrow trajectory = P$$

$$\text{each image in test sequence} \Longrightarrow vertex = p_i$$

Let u denote the vector from x to p. The point p projects onto the segment v at p' (Figure 42). Let α denote the position of p' on the vector v.

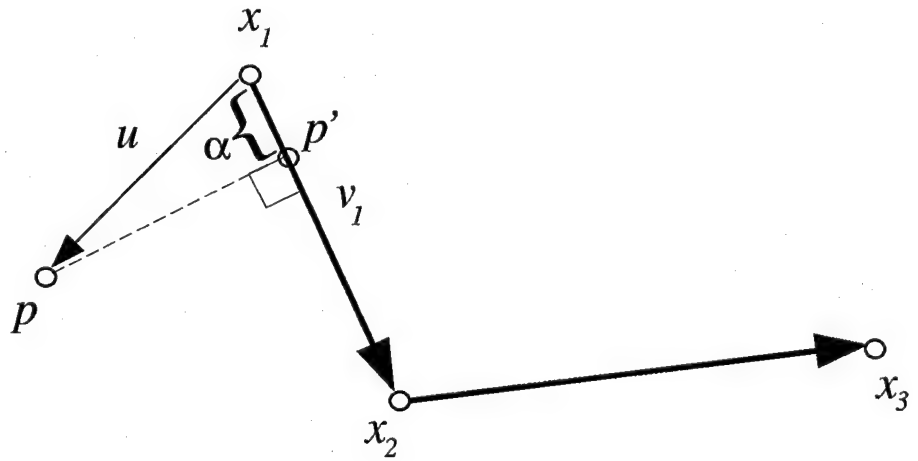

Figure 42. Projection of p onto vector v_1.

$$\alpha = u \cdot v$$

If α is negative, the point p' does not fall on the segment and the distance is calculated from p to x.

$$d = \|x_i - p\|$$

If α is positive but less than l, then p' falls on the segment and the distance is calculated from p to p'.

$$d = \|p - p'\|$$

$$d^2 = \|p - p'\|^2$$

$$d^2 = (p - p') \cdot (p - p')$$

$$d^2 = (p - \{\underbrace{(1 - \frac{\alpha}{l_1})}_{a} x_1 + \underbrace{\frac{\alpha}{l_1}}_{b} x_2\}) \cdot (p - \{(1 - \frac{\alpha}{l_1})x_1 + \frac{\alpha}{l_1}x_2\})$$

$$d^2 = (p - ax_1 - bx_2) \cdot (p - ax_1 - bx_2)$$

$$d^2 = p \cdot p - ap \cdot x_1 - bp \cdot x_2 - ap \cdot x_1 + a^2 x_1 \cdot x_1 + abx_1 \cdot x_2 - bp \cdot x_2 + abx_1 \cdot x_2 + b^2 x_2 \cdot x_2$$

$$d^2 = p \cdot p - 2ap \cdot x_1 - 2bp \cdot x_2 + a^2 x_1 \cdot x_1 + 2abx_1 \cdot x_2 + b^2 x_2 \cdot x_2$$

Using the precalculated inner products.

$$d^2 = p \cdot p - 2ap \cdot x_1 - 2bp \cdot x_2 + a^2 c_{1,1} + 2abc_{1,2} + b^2 c_{2,2}$$

But if α is larger than l, p' does not fall on the segment and the closest part of this segment is the endpoint which will be considered with the next segment.

Check each segment of the trajectory keeping the minimum distance value. Note the segment of the trajectory with the minimum distance because this holds aspect information.

Repeat the process for each point of the test trajectory. The sum distance (Figure 41) is found to each trajectory of the training set. The closest one determines the class. The segment of the training trajectory where each distance projection falls holds the aspect information.

Appendix B. Feature Space Trajectory (FST) Neural Network (NN) User's Manual

B.1 Introduction

This manual applies to the FST NN algorithm written by Gary W. Brandstrom as installed on the MHPCC. This algorithm is designed to run in Matlab version 4.2c, and Unix, but may work on other Matlab versions as well. The user must have a basic knowledge of Matlab and know how to start a Matlab script or function.

This FST NN is designed to detect satellite anomalies from image sequences. The algorithm is designed for low Earth satellites in sun-synchronous orbits. The image sequences must contain the same number of frames and have the same time between frames. Also the images must be collected in a consistent manner (i.e., highest part of pass, same telescope mount and swivel).

B.2 Setup

All the data and Matlab m-files are in a directory named `FSTNN*`. The `'*'` represents a satellite identification code to identify which satellite data occupies the `FSTNN*` directory and it can be a number or a name (Figure 43). Some Matlab m-files not directly executed by the user are located in the user's `matlab` directory. One `FSTNN*` directory contains a database of image sequences of one particular satellite, and is designed to test new image sequences of that satellite. If another satellite is to be tested, the `FSTNN*` directory and all subdirectories must be copied and loaded with the new image sequence database.

The database must be loaded with images. These must be in a Matlab readable format. Tiff files (uncompressed or with packbits compression), or Khoros files can be converted using `tconvert.m` or `kconvert.m`. These scripts can be modified to convert GIF, PCX, or BMP images to Matlab format. Binary files or unix commands can be added to and executed from the Matlab scripts if the user needs to convert images from other formats.

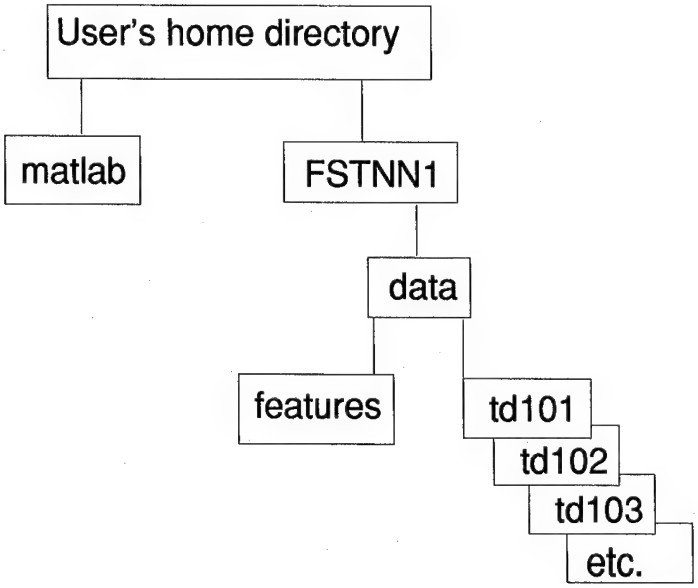

Figure 43. Directory structure for the Matlab FST NN.

B.3 FST NN Procedure

The procedure is broken down into two major parts — training and testing. Training is performed in advance, and takes some time and effort. Testing is a one step process and is very quick and easy.

B.3.1 Training.

1. **Load the normal image sequences (training set) into the database.** — Place the images for each sequence in a numbered `data/td*` subdirectory. The `tconvert.m` script will convert tiff images and `kconvert.m` script will convert khoros images to Matlab format. Run these conversion scripts from the `data/td*` directory where the images are located. The tiff images must have sequentially numbered filenames. After starting Matlab, type 'help kconvert' or 'help tconvert' for assistance. The matlab file will be called `motion.mat` with the variables '`F1, F2, F3, ...`' for each image.

2. **Get features from the training set images.** — Run the `tfeatures.m` script. This is a batch job and the number of the first and last directory of the training set must be entered, with a text editor, before running this script.

3. **Train the FST NN.** — Run `train.m`. This file must be edited to set the number of the first and last directory in the training set. These are the variables 'lo' and 'hi'. This is a batch job that must include all the training database sequences. The script is set up to perform the FST NN distance test using 8 wedge features and the leave-one-out classification threshold.

   Option: The code for the sequence test is in the script but commented out. If the user wishes to implement the sequence test, each training and test sequence must have a consistent number of images.

   Option: A map can be displayed that geographically shows the passes in the training set. This is so the user can determine if there are significant gaps in the training database. In order for this option to be used the following information must be included in a file named `data/td*/motion.out`: latitude, longitude, TGT lat. The latitude and longitude numbers must be the first two fields on the line and the text 'TGT lat' can occur anywhere on the line. For example,

   45.43 205.3 TGT lat TGT lon

   The map is then saved to a file named `map.eps`.

B.3.2 Testing.

1. **Load the image sequence to be tested into the database.** — Place the image sequence in a directory numbered 'td201' or higher. Convert to matlab format if necessary.

2. **Test the image sequence** — Run `fstnn.m`. You will be prompted for the three digit number for the directory where the test image sequence is stored. Classification results will be displayed.

3. **Interpret the results** — The results not only give the normal/anomalous classification, but also show how close the results are to the threshold. If the option is enabled and the `motion.out` files exist, a map is displayed showing where the test pass is located with respect to the training passes. If it falls in a large gap, there is a good chance of being incorrectly classified as an anomalous pass.

B.4 Files and directories used:

- `Directory`
  - `Filename` — Description
- `matlab`
  - `dupstop.m` — This function eliminates duplication in features as they are retrieved. This is necessary because the `fst_trn.m` function will not run if duplicate features exist in a sequence.
  - `fst_trn.m` — This function takes the features from the training set and precalculates some of the values that will be used in `fstnn.m`.
  - `fst_tst.m` — This function performs the FST NN distance calculations.
  - `fst_tst2.m` — This function performs the FST NN distance calculation on the closest trajectory and returns the sequence information.
  - `tconvert.m` — This function converts images from tiff to matlab
  - `wedge.m` — This m-file calculates the 8 wedge feature vector. The 2D DFT is calculated, the values for the 8 wedges are obtained, then the feature vector is energy normalized.
- `FSTNN*`
  - `fstnn.m` — This is the basic FST NN algorithm that tests an image sequence and classifies it as normal or anomalous.

- `tconvert.m` — Converts a sequence of images from tiff to matlab format.
- `kconvert.m` — Converts a sequence of images from khoros viff to matlab format.
- `pfeature.m` — Calculates features for test set. Can work in batch or one image sequence at a time.
- `tfeatures.m` — Calculates features for training set. Can work in batch or one image sequence at a time.
- `train.m` — A batch training algorithm. Loops through the training feature data, executes `fst_trn.m`, and calculates thresholds for the distance and sequence tests. If latitude and longitude information is available, a plot of the satellite passes can be produced.

- `FSTNN*/data/td###` — these are the directories where all the image sequences are stored. Each sequence is in a separate directory named "`/td###`" where the numbers are sequential for the training set (known normal sequences) starting with 101. For the test set, the numbers start with 201 and do not need to be sequential unless a batch of data is tested simultaneously. The file containing the image sequence is called `motion.mat`.

- `FSTNN*/data/features` — this directory holds the training and test set feature vectors. Each training sequence is named `T###.mat` and each test set is named `P###.mat`. The numbers correspond to an image sequence from `td###` with the same number.

- `FSTNN*/data/wedg8.mat` — this file holds the template for the 8 wedge features.

B.5 Matlab m-files (alphabetical)

- **compress.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The script file is              ../matlab/compress.m
%
% This script will compress images by making each pixel value an integer.
% The images must be in variables F1, F2, etc., and the variable 'n' has
% the number of images.  This script doesn't load or save.
```

```
for k=1:n,
  eval(['F' int2str(k) '=round(F' int2str(k) ');'])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **dupstop.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The function file is                    ../matlab/dupstop.m
% usage                                    [Fnew]=dupstop(F)
%
% variables F = feature matrix, each row is a feature vector
%     representing an image.
% Fnew = same as F is there are no duplicate feature
%     vectors in the sequence
%
% This function eliminates duplication in features as they are
% retrieved.  This is necessary because the fstnn training script will
% not run if duplicate features exist in a sequence.

function [Fnew]=dupstop(F)

D=diff (F);
s=sum(D');
n=length (s);

Fnew=F(1,:);
for i=1:n,
  if s(i)==0,
  ['Warning --- a duplicate feature was removed']
  ['Sequence length will vary.  Use train2 and fstnn2']
  else, Fnew=[Fnew;F(i+1,:)];
  end  % if
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **fst_trn.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The function file is                    ../matlab/fst_trn.m
% usage                                    [c,v,len]=fst_trn (T)
%
%%%%%%%%%%%%%%%%%%%%%%%%  variables in %%%%%%%%%%%%%%%%%%%%%%%%
%
%  T =  a matrix where each row represents a feature vector
% from an image sequence.
%
%%%%%%%%%%%%%%%%%%%%%%%%  variables out  %%%%%%%%%%%%%%%%%%%%%
```

```
%
%  c =  inner product matrix of T*T', where VIPs are held
%
%  v =  matrix where each row represents a segment between
%   points on a trajectory
%
%  len= length of segment between points on a trajectory
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [c,v,len]=fst_trn (T)     %% -- to train fstnn

[s,dim]=size(T); % s = number of points (samples) in trajectory,
% dim = length of feature vector
c=T*T';

for i=1:s-1,
  temp=T(i+1,:)-T(i,:);  % this is (x2-x1)
  v(i,:)=temp/sqrt(temp*temp');
  len(i)=sqrt(temp*temp'); % this is ||x2-x1||
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **fst_tst.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The function file is                    ../matlab/fst_tst.m
% usage                          [D,Pp] = fst_tst (c,v,len,T,P,ss)
%
%%%%%%%%%%%%%%%%%%%%%%%%  variables in %%%%%%%%%%%%%%%%%%%%%%%%
%
%  c =  inner product matrix of T*T', where VIPs are held
%
%  v =  matrix where each row represents a segment between
%   points on a trajectory
%
%  len= length of segment between points on a trajectory
%
%  T =  a matrix where each row represents a feature vector
% from a known image sequence.
%
%  P =  a matrix where each row represents a feature vector
% from a unknown image sequence.
%
%  ss = number in test sequence
%
%  s  = number in training sequence
%
%%%%%%%%%%%%%%%%%%%%%%%%  variables out  %%%%%%%%%%%%%%%%%%%%%%
%
%  D =  vector representing distance from each point of P to
```

```
% the trajectory
%
%  Pp = nearest point on trajectory where each point P projects
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [D,Pp] = fst_tst (c,v,len,T,P,ss)

[s,dim]=size(T); % s = number of points (samples) in trajectory,
% dim = length of feature vector
Pptemp = zeros(s,dim);  %initialize matrix

for n=1:ss,   % 1:s loop once for each test point (P)
  u=T(1,:)-P(n,:); % distance of nth test point to first point of trajectory
  d(n,1)=sqrt(u*u');
  Pptemp(1,:)=T(1,:); %% Pptemp is P prime

%%% next find distance of nth test point to each segment
    ef=T*P'; %% e is (i,n) and f is (i+1,n)
 for i=1:s-1;  % for each segment
    u=P(n,:)-T(i,:);
    alpha=u*v(i,:)';

    if alpha<=0,
      d(n,i+1)=realmax; %realmax;
    elseif alpha>len(i),
      temp=P(n,:)-T(i+1,:);   %% new temp is (P-x2)
      d(n,i+1)=sqrt(temp*temp');
      Pptemp(i+1,:)=T(i+1,:);
    else,
      a=1-alpha/len(i);
      b=alpha/len(i);
      d(n,i+1)=P(n,:)*P(n,:)' - 2*a*ef(i,n) - 2*b*ef(i+1,n) ,...
+ 2*a*b*c(i,i+1) + a*a*c(i,i) + b*b*c(i+1,i+1);
      d(n,i+1)=sqrt(d(n,i+1));
      Pptemp(i+1,:)=a*T(i,:)+b*T(i+1,:);
    end
  end

[D(n),ind]=min(d(n,:)); % min distance from P to trajectory
Pp(n,:)=Pptemp(ind,:); % point on traj corresp to min dist
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **fst_tst2.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The function file is                ../matlab/fst_tst2.m
% usage                     [S,D,Pp,d] = fst_tst2 (c,v,len,T,P,ss)
% This is the same as fst_tst.m except it returns sequence info
%%%%%%%%%%%%%%%%%%%%%%  variables in %%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
%  c =  inner product matrix of T*T', where VIPs are held
%
%  v =  matrix where each row represents a segment between
%   points on a trajectory
%
%  len= length of segment between points on a trajectory
%
%  T =  a matrix where each row represents a feature vector
% from a known image sequence.
%
%  P =  a matrix where each row represents a feature vector
% from a unknown image sequence.
%
%  ss = the number of images in a sequence
%
%  s  = number in training sequence
%
%%%%%%%%%%%%%%%%%%%%%%  variables out  %%%%%%%%%%%%%%%%%%%
%
%  D =  vector representing distance from each point of P to
% the trajectory
%
%  Pp = nearest point on trajectory where each point P projects
%
%  S =  vector representing sequence of line segment projections %*
%
%this version is just run for the winning trajectory
%added lines identified by %*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [S,D,Pp,d] = fst_tst2 (c,v,len,T,P,ss)

[s,dim]=size(T); % s = number of points (samples) in trajectory,
% dim = length of feature vector
Pptemp=zeros(s,dim);  %initialize

for n=1:ss,   % loop once for each test point
  u=T(1,:)-P(n,:);  % distance of nth test point to first point of trajectory
  d(n,1)=sqrt(u*u');
  Pptemp(1,:)=T(1,:); %% Pptemp is P prime

%%%   now find distance of nth test point to each segment
    ef=T*P'; %% e is (i,n) and f is (i+1,n)
  for i=1:s-1;  % for each segment
    u=P(n,:)-T(i,:);
    alpha=u*v(i,:)';

    if alpha<=0,
      d(n,i+1)=realmax; %realmax;
    elseif alpha>len(i),
      temp=P(n,:)-T(i+1,:);   %% new temp is (P-x2)
```

```
    d(n,i+1)=sqrt(temp*temp');
    Pptemp(i+1,:)=T(i+1,:);
   else,
    a=1-alpha/len(i);
    b=alpha/len(i);
    d(n,i+1)=P(n,:)*P(n,:)' - 2*a*ef(i,n) - 2*b*ef(i+1,n),...
+ 2*a*b*c(i,i+1) + a*a*c(i,i) + b*b*c(i+1,i+1);
    d(n,i+1)=sqrt(d(n,i+1));
    Pptemp(i+1,:)=a*T(i,:)+b*T(i+1,:);
   end
  end

[D(n),ind]=min(d(n,:)); % min distance from P to trajectory
Pp(n,:)=Pptemp(ind,:); % point on traj corresp to min dist
S(n)=ind; % segment where Pp projects       %*****
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **fstnn.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The script file is                    ./fstnn.m
%
% This is the feature space trajectory neural net (fstnn) algorithm
% that tests an image sequence and classifies it as normal or anomalous.
% Before this script is executed, features must be available
% in directory 'data/features' and training must have been
% accomplished.
%       ------------------------------------
%  THIS IS SET UP FOR THE FST NN DISTANCE TEST WITH LEAVE-ONE-OUT THRESHOLD.
%  Sequence test may be implemented by changeing this and train.m
%
%  this script calls the functions 'fst_tst.m' and 'fst_tst2.m'
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
cd data
load range % this tells the algorithm what is the range of training image directories
cd features
['What sequence would you like to test?']
j = input('Enter the three digit number --- ');
eval(['load P' int2str(j)])   % load test sequence, features in matrix P
[ss,dim]=size(P); % each row of P represents an image feature vector

for i=lo:hi, % this will repeat for each training trajectory, set this!!!!!!
  eval(['load T' int2str(i)]) % this loads trained data, that was stored in
% files T1, T2,... each with variables T,c,v,& len that will run
% against the test sequence P
  [D,Pp] = fst_tst (c,v,len,T,P,ss);          %****
  Dist(i-100)=mean(D); % this variable will hold total distance to each training
```

```
end % sequence

[cDist,k]=min(Dist); % this will find which trajectory was closest

%%%%% this stores the distance ****
results=[results;cDist];

% now a decision must be made on whether these numbers are acceptable
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%   this section runs the sequence test.  If sequences are various lengths
%%   comment out the following lines
%%
% Reload that trajectory to test sequential match
%eval(['load T' int2str(k+100)])
%[S,D,Pp,d] = fst_tst2 (c,v,len,T,P,ss);          %****

%%%%% this stores the sequence ****
%Seq=[Seq;S];


%best=1:ss;  %
%for k=1:length(results),
%  Sdif(k,:)=Seq(k,:)-best;
%end
%result2=std(Sdif');   % sequence test results 'result2' would be compared
% to SeqSt instead of DistSt

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% this makes results show up
load DistSt
['****************************']
if results < 1.3 * mean(DistSt),  % <<<====== MAY ADJUST THE THRESHOLD HERE
  text=['********** Normal ***********']
else,
text = ['******* Anomalous *********']
end
['****************************']
threshold =  1.3 * mean(DistSt)
results
eval(['save tstrslt' int2str(j) ' results text'])
cd ../..
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                  The following section is optional
%%  If a map was produced in train.m, then you can produce a
%%  map here to help evaluate the results.  If the test images come
%%  from a gap in the training data, this will show up here.
```

- **kconvert.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%script kconvert.m   ***** to convert khoros movie files to matlab  ******
%
% User must edit the mfile to set up the image filename.
% Execute the script in the directory where the khoros file is located.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

!vconvert -i atn67477_M_0002 -o temp -t "float" -b 0 %<<===EDIT FILENAME HERE
!/apps/Khoros/toolboxes/osutools/bin/viff2mtlb -i1 temp -o movie.mat ,...
-num_inputs 1

x=size(movie,1);

clear
load movie

if x==462,
  movie=movie(104:359,:);
end
clear x;

%%%% breakup movie into frames
%%%% assumes images are 128x128
n=length (movie) / 128;

for i=1:n;
  j=i*128;
  eval(['F' int2str(i) ' = movie(129:256,j-127:j);']);
end

%%%F1 will be variable name for frame 1; F2 for frame 2, ..., etc

clear i j temp movie
save motion
cd .. % back out of dir

end
```

- **pfeature.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The script file is                          ./pfeature.m
%
% This is an algorithm that calculates features for the test set.
% It can be used on individual image sequences by setting the low
% and hi numbers in the mfile.
%
%       ------------------------------------
%  THE USER MUST SET THE LOW AND HI NUMBERS OF THE IMAGE
%  DIRECTORIES TO BE PROCESSED BY EDITING THIS MFILE
%       ------------------------------------
```

```
%
%  How it works:
%  Images to be processed must be in directories named data/td201,
%  td202, td203, ....
%  The test data numbers range from 201 to 999 and do not need to be
%  used sequentially unless the features are calculated in batch.
%  The images should be saved in a file named 'motion.mat' with the
%  variable names F1, F2, ..., F20 --- one variable per image.
%  The variable 'n' in motion.mat denotes the number of images in the
%  sequence.
%  The calculated features are saved in the directory 'data/features'
%  in files P201, P202, etc., matching the file number of the image
%  directories (i.e. td201).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
lo=251              %-----------------------------------
hi=255 %      <<==== SET THESE TO COVER TRAINING SET RANGE!!!!
                    % -----------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=lo:hi,
dirname=['data/td' int2str(i)];
eval(['cd ' dirname])
load motion
cd .. %back to 'data' dir%
clear x
  for k=1:n, %for each image, get wedge features%
  eval(['F(k,:)=wedge(F' int2str(k) ');'])
  eval(['clear F' int2str(k)])
  end
[P]=dupstop (F);

cd features
eval(['save P' int2str(i)  ' P'])
clear P T1 T F
cd ../..
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **tconvert.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%script tconvert.m   ***** to convert tiff files to matlab  ******
%
% User must edit the mfile to set up the image filenames.
% Execute the script in the directory where the tiff files are located.
% This only works with tiff files that are uncompressed or packbits
% compressed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
clear
lo=1              %-----------------------------------
hi=20; %      <<==== SET THESE TO COVER SET RANGE of images!!!!
                 % -----------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


x=['td81'];  %input('Name the sat:  ','s');%<== EDIT THIS part
% of filename 'td81' ?
for i=lo:hi,
  if i==10, break
  end

                                              % EDIT THIS!! '.p.tif'
  eval(['[F' int2str(i) ',map]=tiffread(''' x '.00' int2str(i) '.p.tif'');'])
end

if i==10,
  for i=10:hi
                                              % EDIT THIS!! '.p.tif'
  eval(['[F' int2str(i) ',map]=tiffread(''' x '.0' int2str(i) '.p.tif'');'])
  end
end

clear map i xx
%deci   % script to decimate image by 1/4 -- saves storage space
clear h i j k w
save motion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **tfeature.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The script file is                          ./tfeature.m
%
% This is a batch algorithm that calculates features for the
% training set.  It can be used on individual image sequences
% by setting the low and hi numbers in the mfile.
%
%       -----------------------------------
%  THE USER MUST SET THE LOW AND HI NUMBERS OF THE IMAGE
%  DIRECTORIES TO BE PROCESSED BY EDITING THIS MFILE
%       -----------------------------------
%
%  How it works:
%  Images to be processed must be in directories named data/td101,
%  td102, td103, ....
%  The training data numbers range from 101 to 199 and must be
%  used sequentially.  The images should be saved in a file named
% 'motion.mat' with the variable names F1, F2, ..., F20 ---
```

```
%  one variable per image.  The variable 'n' in motion.mat
%  denotes the number of images in the sequence.
%  The calculated features are saved in the directory 'data/features'
%  in files T101, T102, etc., matching the file number of the image
%  directories (i.e. td101).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
lo=101              %-----------------------------------
hi=105 %      <<==== SET THESE TO COVER TRAINING SET RANGE!!!!
                    % -----------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=lo:hi,
dirname=['data/td' int2str(i)];
eval(['cd ' dirname])
load motion
cd .. %back to 'data' dir%
clear x
  for k=1:n, %for each image, get wedge features%
  eval(['F(k,:)=wedge(F' int2str(k) ');'])
  eval(['clear F' int2str(k)])
  end
[T]=dupstop (F);

cd features
eval(['save T' int2str(i)  ' T'])
clear T1 T F
cd ../..
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **train.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The script file is                          ./train.m
%
% This is a batch training algorithm that calculates some values
% needed for the fstnn.  Features from the mat files T1, T2, ...
% are loaded, processed and saved to the same files.
%       -----------------------------------
%  THE USER MUST SET THE LOW AND HI NUMBERS OF THE 'T'
%  FEATURE FILES BY EDITING THIS MFILE
%       -----------------------------------
%
% This is set up for the FST NN distance test and leave-one-out threshold
% Sequence test may be implemented by editing this script and 'fstnn.m'
%
%  this script calls the functions 'fst_trn.m' and 'fst_tst.m'
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear
cd data

lo=101         %------------------------------------
hi=105; %       <<==== SET THESE TO COVER TRAINING SET RANGE!!!!
                      % ------------------------------------
save range hi lo
cd features
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   This section is to precalculate some variables needed for
%   testing %

for i=lo:hi,
  eval(['load T' int2str(i)])  % loads feature matrix
  [c,v,len]=fst_trn (T);
  eval(['save T' int2str(i)  ' T c v len'])
  clear T c v len
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  This section does the leave-one-out test so
%  a classification boundary can be found
%


for j=lo:hi,
eval(['load T' int2str(j)])   % load test sequence, features should be in matrix P
% each row of P is an image feature vector
P=T; % this is if test features were saved as variable T


[ss,dim]=size(P);

for i=lo:hi, % this will repeat for each training trajectory

if i==j, % this will cause a set to skip itself
Dist(i-100)=realmax;
else,

  eval(['load T' int2str(i)]) % this load trained data, stored in
% files T1, T2,... each with variables T,c,v,& len that
% will run against the test sequence P

  [D,Pp] = fst_tst (c,v,len,T,P,ss);          %****


  Dist(i-100)=mean(D); % this variable will hold total distance to each
% training sequence
```

```
end %if
end %i loop

[cDist,k]=min(Dist); % this will find which trajectory was closest

title=['Testing td' int2str(j) ' ************************************************************']

%%%%%% this calculates the sequence info ****
% reload that trajectory to test sequential match
%eval(['load T' int2str(k)])
%[S,D,Pp,d] = fst_tst2 (c,v,len,T,P,ss);          %%%%%
%Seq=[Seq;S];

results=[results;cDist];
end

%%%%%%%%% Sequence test %%%%%%%%
%best=1:ss;
%for k=1:23,
%  Sdif(k,:)=Seq(k,:)-best;
%end

%result2=std(Sdif');

DistSt=results;
%SeqSt=result2;

save DistSt DistSt
%save SeqSt SeqSt
cd ../..

%OPTIONAL:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  This section produces the optional map of training set passes.
%  If lat long data is not available, comment out this section,
%  or make backup and delete this section
%cd data
%clear i
%close
%aplot=zeros(20,2); %<<== THIS IS FOR INITIALIZING A 20 IMAGE SEQUENCE

%for i=lo:hi,
%  eval(['cd td' int2str(i)])
%  unix('grep ''TGT lat,'' motion.out >../aplot.all');
%  cd ..
%  unix('awk ''{print $1,$2}'' aplot.all >aplot.num');
%  load aplot.num
%  A=[A;aplot];
%end

%plot (A(:,2),A(:,1),'.')
```

```
%axis([-180,-135,5,35])
%hold
%maui=[20.7 -156.3];
%plot (maui(:,2),maui(:,1),'*')
%grid
%xlabel('longitude')
%ylabel('latitude')
%legend('*','Maui','.','Satellite position')
%x=['First ' int2str(i-100) ' passes'];
%title(x)
%cd .. % back to fstnn directory
%print map -deps  %This saves the map to file 'map.eps'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **wedge.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function file:                        ../matlab/wedge.m
% usage:                                [feat]=wedge(F);
%
%       output:                         feat = feature vector
%        input:                         F = frame of imagery (array)
%
% function for wedge sampled DFT feature vector, this version does 8
% wedges, left half and uses the template '8wedg.mat' to speed processing
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% variables
% feat =feature vector out
% F =frame of imagery (array) in, (becomes dft left half)
% h =height of frame
% w =width of frame
% r =max radius of values included in temp sum
% temp =vector summing values of fft within wedge
% count =vector counting number of values summed
% i,j =generic counters
% n =number of wedges
% we =wedge width in radians
% wed =vector of wedge bins
% template# = wedge templates loaded from 'data/wedg8.mat'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[feat]=wedge(F);

%%%%%%%   load desired wedge templates here   &&&&&&&
load wedg8

%feat=zeros(1,n+1);
feat=zeros(1,n); %initialize

%% make fourier transform matrix %%
```

```
F=fftshift(abs(fft2(F)));
tempdc = F(hh+1,w+1); % get dc value, just in case it is needed
F=F(:,1:w);                % cut fft image in half

%%%%%%%%  find total values in each wedge bin  %%%%%%%
for i=1:n,
  eval(['temp=F.*template' int2str(i) ';'])
  feat(i)=sum(sum(temp))/count(i);
end

%%%%%%%%   add feature for dc?    %%%%%%%%%%%%%%%
%% note -- for even length and width, the dc value is located at
%%      length/2 + 1 and width/2 + 1, for odd it's the middle
%
%feat(n+1)=tempdc/mean(count);  %% <<==IF YOU WANT DC, include this line.

%% now normalize feature vector, if desired
tot=sqrt(sum(feat.^2'))';
n=length(tot);
  for i=1:n,
  feat(i,:)=feat(i,:)/tot(1,:);
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Bibliography

1. Bate, Roger R, et al. *Fundamentals of Astrodynamics*. Dover Publications, Inc., 1971.
2. Buglia, James J. *Compilation of Methods on Orbital Mechanics and Solar Geometry*. reference publication 1204, NASA, October 1988.
3. Duda, R. O. and P. E. Hart. *Pattern Recognition and Scene Analysis*. Wiley, New York, 1973.
4. Entropic Research Laboratory, Inc., "Hidden Markov Model Toolkit (HTK)," 1992.
5. Fielding, Kenneth H. *Spatio-Temporal Pattern Recognition using Hidden Markov Models*. PhD Dissertation, Air Force Institute of Technology, June 1994.
6. Fielding, Kenneth H. and Dennis W. Ruck. "Spatio-Temporal Pattern Recognition using Hidden Markov Models," *IEEE Transactions on Aerospace and Electronic Systems* (1995). Accepted for publication.
7. Fisher, R. A. "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, *7*:179–188 (1936).
8. Hamanadi, Naser A. *Automatic Target Cueing in IR Imagery*. Masters Thesis, Air Force Institute of Technology, 1981.
9. Hand, Maj Richard, et al., editors. *Space Handbook: An Analyst's Guide*, *2*. Air University Press, Maxwell Air Force Base, Alabama, December 1993. prepared by Maj Michael J. Muolo.
10. Lee, Chulhee and David Landgrebe. "Decision boundary feature extraction for nonparametric classification," *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(2):433–444 (March 1993). Lee and Landgrebe - nonparametric.
11. Levinson, S. E., et al. "An introduction to the applications of the theory of probabilistic functions of a Markov process to automatic speech recognition," *The Bell System Technical Journal*, *62*(4):1035–1074 (1983).
12. Neiberg, Leonard and David. P. Casasent. "Feature space trajectory (FST) classifier neural network," *Proc. Soc. Photo-Opt. Instrum. Eng. (SPIE)*, *2353*:276–292 (1994).
13. Neiberg, Leonard and David P. Casasent. "Feature space trajectory neural network classifier," *Proc. Soc. Photo-Opt. Instrum. Eng. (SPIE)*, *2492*:361–372 (1995).
14. Poritz, Alan B. "Hidden Markov Models: A guided tour." *Proceedings of the ICASSP*. 7–13. 1988.
15. Rabiner, L. R. and B. H. Juang. "An Introduction to hidden Markov models," *IEEE ASSP Magazine*, 4–16 (January 1986).
16. Rabiner, Lawrence R. "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, *77*(2):257–286 (1989).

17. Rosenblatt, F. *The perceptron—a perceiving and recognizing automaton.* Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca N.Y., January 1957.

18. The Mathworks, Inc., "Matlab, version 4.0," 1993.

19. Unknown. "Maui AMOS brief." from Phillips Lab, 1994.

Vita

Gary W. Brandstrom was born in El Paso, TX, 1959. Earned a B.S. in Geology from Hardin-Simmons University, Abilene TX, 1982. Worked for Crown Exploration Company, Abilene, TX, 1982-1984. Earned M.S. in Geology from Texas A&M University, 1986. Entered US Air Force in 1986.

First duty assignment in the Air Force was as an imagery intelligence analyst at the 12th Tactical Intelligence Squadron. Was selected to attend Undergraduate Space Training in 1990. Second assignment was at USSPACECOM/J3 from 1990-94. Next assignment will be at AF Space Command/DO.

Permanent address: 10205 Camwood
El Paso, Texas 79925

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1995	*Master's Thesis*

4. TITLE AND SUBTITLE
SPACE OBJECT IDENTIFICATION Using Spatio-Temporal Pattern Recognition

5. FUNDING NUMBERS

6. AUTHOR(S)
Gary W. Brandstrom, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GSO/ENG/95D-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt Bruce Stribling, Phillips Lab/OL-YY
535 Lippoa Pkwy, Ste 200
Kihei, Maui, HI 96753

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis is part of a research effort to automate the task of characterizing space objects or satellites based on a sequence of images. The goal is to detect space object anomalies. Two algorithms are considered — the feature space trajectory neural network (FST NN) and hidden Markov model (HMM) classifier. The FST NN was first presented by Leonard Neiberg and David P. Casasent in 1994 as a target identification tool. Kenneth H. Fielding and Dennis W. Ruck recently applied the hidden Markov model classifier to a 3D moving light display identification problem and a target recognition problem, using time history information to improve classification results. Time sequenced images produced by a simulation program are used for developing and testing the anomaly detection algorithms. A variety of features are tested for this problem. Features are derived from the two dimensional (2D) discrete Fourier transform (DFT) with various normalization schemes applied. The FST NN is found to be more robust than the HMM classifier. Both algorithms are capable of achieving perfect classification, but when shot noise is added to the images or when the image sample spacing is increased, the FST NN continues to perform well while the HMM performance declines. A new test is presented that measures how well a test sequence matches other sequences in the database. The FST NN is based strictly on feature space distance; but if the order of the sequence is important, the new test is useful.

14. SUBJECT TERMS
Pattern Recognition, Space Object Identification, Spatio-Temporal Analysis, Feature Space Trajectory Neural Net, Hidden Markov Models, Image Processing, Neural Nets

15. NUMBER OF PAGES
97

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL